175 PTAS

66



CURSO PRACTICO DEL ORDENADOR PERSONAL, EL MICRO Y EL MINIORDENADOR



Editorial Delta, S.A.

MI COMPUTER CURSO PRACTICO
DEL ORDENADOR PERSONAL, EL MICRO Y EL MINIORDENADOR

Publicado por Editorial Delta, S.A., Barcelona

Volumen VI-Fascículo 66

Director: José Mas Godayol
Director editorial: Gerardo Romero
Jefe de redacción: Pablo Parra
Coordinación editorial: Jaime Mardones, Francisco Martín
Asesor técnico: Ramón Cervelló

Redactores y colaboradores: G. Jefferson, R. Ford, F. Martín, S. Tarditti, A. Cuevas, F. Blasco
Para la edición inglesa: R. Pawson (editor), D. Tebbutt (consultant editor), C. Cooper (executive editor), D. Whelan (art editor), Bunch Partworks Ltd. (proyecto y realización)

Realización gráfica: Luis F. Balaguer

Redacción y administración:
Paseo de Gracia, 88, 5.°, 08008 Barcelona
Tels. (93) 215 10 32 / (93) 215 10 50 - Télex 97848 EDLTE

MI COMPUTER, *Curso práctico del ordenador personal, el micro y el miniordenador*, se publica en forma de 96 fascículos de aparición semanal, encuadernables en ocho volúmenes. Cada fascículo consta de 20 páginas interiores y sus correspondientes cubiertas. Con el fascículo que completa cada uno de los volúmenes, se ponen a la venta las tapas para su encuadernación.
El editor se reserva el derecho de modificar el precio de venta del fascículo en el transcurso de la obra, si las circunstancias del mercado así lo exigieran.

ISBN: 84-85822-83-8 (fascículo) 84-7598-034-7 (tomo 6)
84-85822-82-X (obra completa)
Depósito Legal: B. 52/1984

Fotocomposición: Tecfa, S.A., Pedro IV, 160, Barcelona-5
Impresión: Cayfosa, Santa Perpètua de Mogoda (Barcelona) 178504

Impreso en España-Printed in Spain-Abril 1985

Editorial Delta, S.A., garantiza la publicación de todos los fascículos que componen esta obra.

Distribuye para España: Marco Ibérica, Distribución de Ediciones, S.A., Carretera de Irún, km 13,350. Variante de Fuencarral, 28034 Madrid.
Distribuye para Colombia: Distribuidoras Unidas, Ltda., Transversal 93, n.° 52-03, Bogotá D.E.
Distribuye para México: Distribuidora Intermex, S.A., Lucio blanco, n.° 435, Col. San Juan Tlihuaca, Azcapotzalco, 02400, México D.F.
Distribuye para Venezuela: Distribuidora Continental, S.A., Edificio Bloque Dearmas, final Avda. San Martín con final Avda. La Paz, Caracas 1010.

Pida a su proveedor habitual que le reserve un ejemplar de MI COMPUTER. Comprando su fascículo todas las semanas y en el mismo quiosco o librería, Vd. conseguirá un servicio más rápido, pues nos permite realizar la distribución a los puntos de venta con la mayor precisión.

Servicio de suscripciones y atrasados (sólo para España)

Las condiciones de suscripción a la obra completa (96 fascículos más las tapas, guardas y transferibles para la confección de los 8 volúmenes) son las siguientes:

a) Un pago único anticipado de 19 425 ptas. o bien 8 pagos trimestrales anticipados y consecutivos de 2 429 ptas. (sin gastos de envío).

b) Los pagos pueden hacerse efectivos mediante ingreso en la cuenta 6.850.277 de la Caja Postal de Ahorros y remitiendo a continuación el resguardo o su fotocopia a Editorial Delta, S.A. (Paseo de Gracia, 88, 5.°, 08008 Barcelona), o también con talón bancario remitido a la misma dirección.

c) Se realizará un envío cada 12 semanas, compuesto de 12 fascículos y las tapas para encuadernarlos.

Los fascículos atrasados pueden adquirirse en el quiosco o librería habitual. También pueden recibirse por correo, con incremento del coste de envío, haciendo llegar su importe a Editorial Delta, S.A., en la forma establecida en el apartado b).

Para cualquier aclaración, telefonear al (93) 215 75 21.

No se efectúan envíos contra reembolso.





El nombre del robot

Examinemos algunos de los productos que se venden comercialmente bajo la denominación de robot

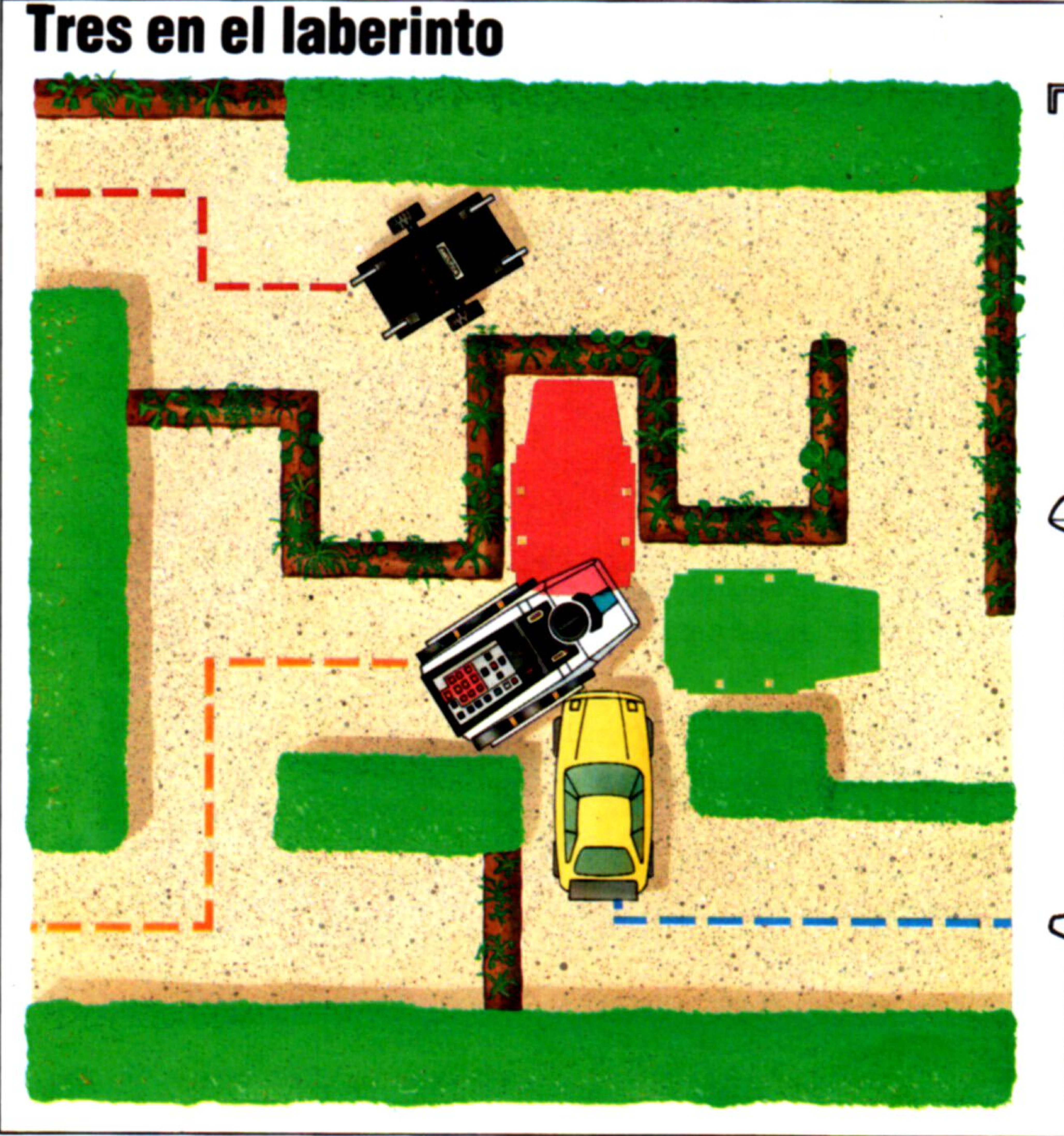
Hasta ahora en nuestra serie sobre Robótica nos hemos ocupado fundamentalmente de consideraciones teóricas relativas al diseño y el funcionamiento de los robots. En la práctica, no se han implementado muchos de los conceptos que hemos analizado, o se los ha limitado debido a una cierta falta de componentes mecánicos y/o de software inteligente. Los robots actuales, ya sea los destinados al uso doméstico o al uso industrial, tienden a quedarse cortos en relación a lo que hemos llegado a esperar de ellos con el paso de los años. Existen sensores para dotar al robot de vista, oído y tacto, pero las "sensaciones" que éste experimenta no significan nada para él, y no se las puede sintetizar para estimularlo hacia un comportamiento original, no programado. Robbie el Robot y sus otros compañeros de ficción todavía están muy lejos de hacerse realidad.

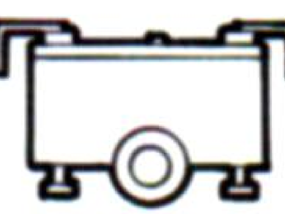
No obstante, en la actualidad se están vendiendo muchos productos bajo la denominación común de *robot*. Éstos van desde pequeños y baratos juguetes hasta el sofisticado R2D2 y robots industriales sumamente costosos. Después de haber examinado los componentes del diseño de robots y los aspectos teóricos, debemos ahora considerar qué constituye un auténtico robot. No hemos de ser demasiado exigentes, pero estamos en condiciones de reunir todo cuanto sabemos y aplicar estos conocimientos para elaborar una definición funcional.

La primera consideración, que viene a eliminar muchos de los productos "robot" de precio más reducido, es el movimiento: ¿puede el robot desplazarse solo a través de un espacio? No podemos esperar que el robot se programe a sí mismo, ni que establezca un curso de acción sin la guía del hombre; pero sí podemos esperar que, una vez puesto en movimiento, sea capaz de operar con independencia del control humano continuo. Sin esta libertad de movimiento no se puede considerar que un objeto sea un robot.

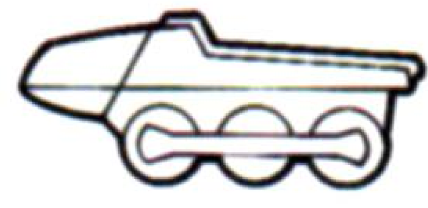
Habiendo superado la prueba del movimiento, nuestro candidato a robot debe evaluarse sobre la base de cómo se efectúa la acción. A un pequeño coche de juguete se le puede instalar un motor y unas pilas que lo mantengan en movimiento siguiendo una línea recta. Agréguele parachoques y el coche podrá desviarse apartándose de obstáculos tales como paredes y mesas. Póngale al coche un centro de gravedad ligeramente inusual y grandes neumáticos de goma y podrá incluso ascender por

Tres en el laberinto





Robot
Alimentado y controlado desde su ordenador madre, el robot está equipado con sensores sensibles al tacto y a la luz



Big Trak
Este dispositivo activado por microprocesador se puede programar a través de su teclado con instrucciones de distancia y dirección similares a las del LOGO



Coche con parachoques
Este juguete a pilas avanzará en línea recta hasta chocar contra algún objeto, en cuyo caso su trayectoria variará sin rumbo fijo

Steve Cross

Pistas sorprendentes
Los tres dispositivos están intentando recorrer un laberinto: el coche de juguete simplemente avanza tropezando de pared en pared, el Big Trak sigue las instrucciones que ha programado su operador humano para superar el laberinto, mientras que nuestro robot se aprende el laberinto a través de la interacción de su software y sus sensores. Al final, el robot conseguirá resolver el laberinto, independientemente de lo que suceda; el Big Trak seguirá su programa, de modo que, en caso de que las directrices del operador sean correctas, podrá superarlo; el coche de juguete sólo podría resolver laberintos "diestros" y, aun así, sería por casualidad. Cuando el coche choca con el Big Trak, el hecho no lo afecta, puesto que su comportamiento no tiene ningún propósito determinado; el Big Trak, sin embargo, sufrirá una desviación de 90° en su recorrido (señalado en color verde) pero seguirá girando y avanzando como si aún estuviera en su camino (señalado en rojo). Ambos dispositivos no reaccionan de forma "inteligente" ante este acontecimiento imprevisto; ante esta situación, el robot se comportaría como si se tratara de un aspecto más de un entorno imprevisible

una pared y darse la vuelta, siguiendo luego en otra dirección. El coche se mueve por sí mismo y es independiente de todo control humano, pero ¿se puede decir que sea un auténtico robot? La respuesta, bastante evidente, es "No", pero para nuestro estudio sobre los robots es de fundamental importancia entender por qué esto es así.

El movimiento del robot, como ya hemos visto, se divide en dos categorías: movimiento simple bajo el control de un programa, y movimiento inteligente. En ambos casos la clave es la programación. Nuestro coche activado por motor no puede ser un robot porque no se lo puede programar para que se mueva de diversas formas. Posee un patrón de operaciones establecido incorporado en él mecánicamente, pero no puede responder a instrucciones humanas y no posee medio alguno en virtud del cual un controlador humano pueda generar un movimiento alternativo. Una interesante variación en este sentido es el coche, autocamión u otro objeto activados por motor que se pueda programar perforando un patrón en una ficha. La ficha se lee mecánicamente y el coche sigue el patrón girando hacia la izquierda o la derecha y siguiendo hacia adelante de acuerdo a las instrucciones de la ficha. Según nuestra definición, este tipo de coche sí sería un robot, porque puede ser programado con el software de ficha perforada. La necesidad de un movimiento programable elimina a muchos otros productos pequeños y baratos etiquetados bajo el denominador común de "robot".

Se pueden aplicar otras varias consideraciones. ¿Posee el objeto sensores que le den una entrada proveniente del mundo exterior? ¿Está capacitado para responder a su entorno y crear un modelo interno cambiante? ¿Puede jugar una partida de ajedrez? A nuestro robot se le podría aplicar cualquiera de estos criterios, pero, en el análisis final, el elemento del movimiento programable es esencial.

Los siguientes productos, todos ellos vendidos como robots, se pueden adquirir en los comercios especializados.

NOMBRE: Beasty
TIPO: Brazo-robot
PROGRAMABLE: Sí
PROVISTO DE SOFTWARE: Sí
REALIMENTACION SENSORIAL: Realimentación posicional
DISTRIBUIDORES: Commotion Ltd, 241 Green Street, Enfield, Middlesex EN3 7SJ, Gran Bretaña

Recibe sus instrucciones desde la puerta para el usuario del BBC Micro y se alimenta a partir de la fuente eléctrica auxiliar del ordenador. Se suministra en forma de modelo para armar e incluye tres servomotores. También se acompaña de dos manuales, un folleto de construcción y un manual de operaciones. El robot lleva incorporado su propio sistema operativo (Robol) que permite que el usuario mueva cada uno de los servomotores de forma independiente.

NOMBRE: Hebot II
TIPO: Robot móvil
PROGRAMABLE: Sí
PROVISTO DE SOFTWARE: Sí
REALIMENTACIÓN SENSORIAL: Táctil
DISTRIBUIDORES: Powertran Cybernetics Ltd, West Portway Industrial Estate, Andover, Hampshire, SP10 3NN, Gran Bretaña

Es una tortuga-robot accionada por dos motores CD conectados a dos ruedas. La tortuga se conecta en interface al conector marginal del Sinclair ZX81, si bien los fabricantes aseguran que, mediante la reconexión de algunos cables, se puede hacer que funcione en cualquier micro personal. Viene en forma de modelo para armar con un folleto de construcción. La tortuga posee un lápiz retráctil y cuatro detectores de colisión que proporcionan realimentación táctil. El suministro eléctrico le llega desde el ordenador.





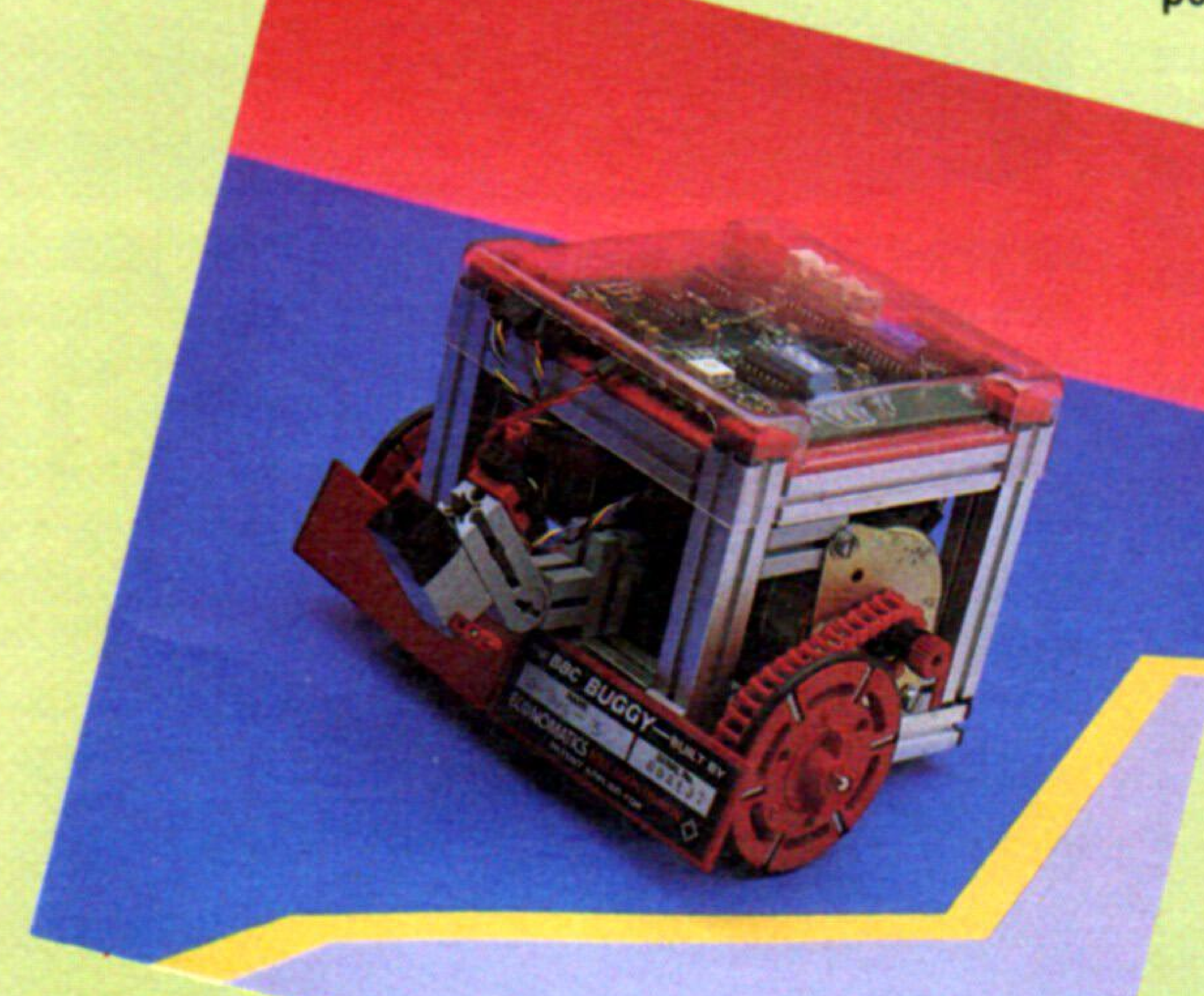
Chris Stevens

NOMBRE: Memocon Crawler
TIPO: Robot móvil
PROGRAMABLE: Sí
PROVISTO DE SOFTWARE: No
REALIMENTACIÓN SENSORIAL: No
DISTRIBUIDORES: Prism Products, Prism House, 18-29 Mora Street, London, EC1, Gran Bretaña

Es el robot más sofisticado de la gama Prism Movits. Emplea pilas de 1,5 V para alimentar dos motores eléctricos CD. El control se efectúa mediante una caja que posee cinco teclas, una para cada una de las instrucciones. Esta caja está conectada al Crawler mediante un cable plano. El dispositivo posee, asimismo, un zumbador y LED (diodos emisores de luz); éstos también se pueden controlar desde el teclado.

NOMBRE: Valiant Turtle
TIPO: Robot móvil
PROGRAMABLE: Sí
PROVISTO DE SOFTWARE: Sí
REALIMENTACIÓN SENSORIAL: Posicional
DISTRIBUIDORES: Valiant Designs, 1st Floor, Park House, Battersea Park Road, London, SW11, Gran Bretaña

Tiene realmente forma de tortuga. Está activado mediante un par de motores; cada uno le proporciona potencia a una única rueda. El dispositivo se controla desde el ordenador mediante un haz infrarrojo. El software que se suministra con el dispositivo está diseñado para ejecutarse mediante LOGO, si bien funcionará igualmente sin el lenguaje de instrucciones. La potencia se la suministra una pila recargable incorporada en el aparato. Tiene un portalápiz que le permite trazar diseños mientras avanza.



NOMBRE: BBC Buggy
TIPO: Robot móvil
PROGRAMABLE: Sí
PROVISTO DE SOFTWARE: Sí
REALIMENTACIÓN SENSORIAL: Sensor luminoso, dispositivos de presión
DISTRIBUIDORES: Economatics Education Ltd, Epic House, 9 Orgreave Road, Handsworth, Sheffield, Gran Bretaña

Es una tortuga controlada mediante software. Viene en forma de modelo para armar y se opera desde la puerta para el usuario del BBC Micro, con potencia proveniente de la fuente de alimentación auxiliar del ordenador. Utiliza un par de motores paso a paso que le proporcionan un movimiento de gran precisión. Puede responder a una fuente luminosa, buscarla y encontrarla; se le pueden instalar un lápiz y un lector de código de barras.



Chris Stevens

NOMBRE: Edinburgh Turtle
TIPO: Robot móvil
PROGRAMABLE: Sí
PROVISTO DE SOFTWARE: Sí
REALIMENTACIÓN SENSORIAL: Posicional
DISTRIBUIDORES: Jessop Acoustics, Unit 5, 7 Long Street, London, E2, Gran Bretaña

A esta tortuga se le ha dado el nombre de Edinburgh en honor a la ciudad en la cual se desarrolló. La Edinburgh Turtle se conecta al ordenador mediante un cable plano, desde donde también obtiene su potencia. La activan un par de motores eléctricos CD; cada uno alimenta a una sola rueda. Está equipada con un portalápiz retráctil y un altavoz incorporado. La firma fabricante acaba de introducir una versión a control remoto.

Conjetura exacta

Concluiremos el estudio del "TK!Solver" con un detenido análisis de algunas de sus características exclusivas

Tal como explicábamos en el capítulo anterior, *TK!Solver* es un paquete de software de la "próxima generación" que introduce el concepto de hoja electrónica en el reino de las matemáticas y la ingeniería de nivel superior. Ya hemos visto que el programa permite que el usuario defina variables con nombres y utilice las mismas en ecuaciones matemáticas complejas. En este capítulo, el último de nuestra serie sobre hojas electrónicas, analizamos con todo detalle la inusual capacidad del *TK!* para *iterar*. Éste es un método en virtud del cual el programa puede resolver una variable haciendo suposiciones respecto a la misma. Normalmente, cuando se trabaja con ecuaciones, uno puede determinar los valores de todas las variables si desde el comienzo se cuenta con la información suficiente. El programa sencillamente reduce el programa a una serie de cálculos. Por ejemplo:

$A^2+B^2=2COS\ Y$

Cualquiera de estas tres variables se puede resolver fácilmente si se conocen los otros dos valores. Enfrentado a esta ecuación (y proporcionándole valores para A y B), el Direct Solver (solucionador directo) del *TK!* llevará a cabo los cálculos requeridos y producirá un valor para Y.

Pero existen ocasiones en las cuales la determinación de un valor no es directa. La ecuación redundante, que define a una variable en términos de sí misma, constituye uno de tales casos. Por ejemplo, tomemos:

D=(A+B)/(2*D)

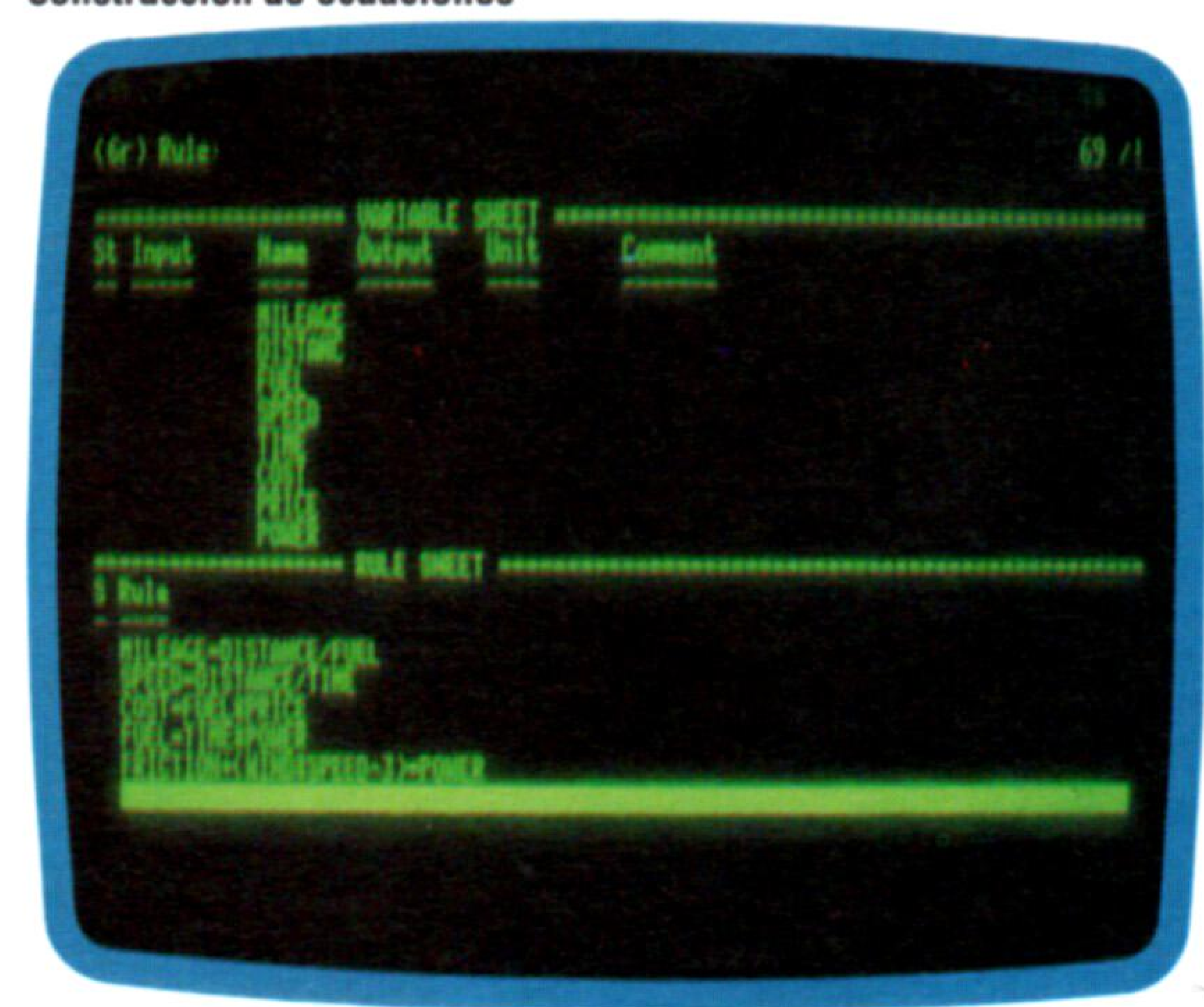
en un modelo en el cual el único valor conocido es A. Se pueden plantear otros problemas a consecuencia de un modelo incompleto, o de un modelo con muchas variables independientes y una cantidad limitada de datos. El concepto de iteración es difícil, de modo que tomemos un ejemplo de iteración que sea más práctico.

Volvamos a considerar el modelo del viaje en coche que creamos en el capítulo anterior y agreguemos unos pocos detalles para hacerlo más aplicable. Como recordará, el modelo previo se construyó alrededor de cinco valores: distancia, tiempo, velocidad, gasolina y gastosviaje. Podría calcular los gastos del viaje, disponiendo de la velocidad y el consumo de gasolina; la distancia, a partir de la velocidad y el tiempo, y algunas otras variaciones sencillas. Pero ¿y si ahora quisiéramos determinar a qué velocidad deberíamos viajar para poder completar el viaje con un presupuesto dado?

En primer lugar, debemos añadir varios factores a nuestro modelo. Por ejemplo, el modelo debería tener en cuenta la potencia del vehículo, la fricción interna del motor y la resistencia del viento, todos los cuales producen un efecto en los gastos de viaje y la velocidad del vehículo. (Supondremos que la fricción interna es constante.) Asimismo, debemos fijar un límite máximo para nuestro presupuesto y el costo del combustible que se esté consumiendo.

Se comienza a construir el verdadero modelo entrando estas ecuaciones en la hoja Rule, una ecuación en cada línea. Estas ecuaciones se leen en la hoja Variable de forma automática. Al haber varias

Construcción de ecuaciones

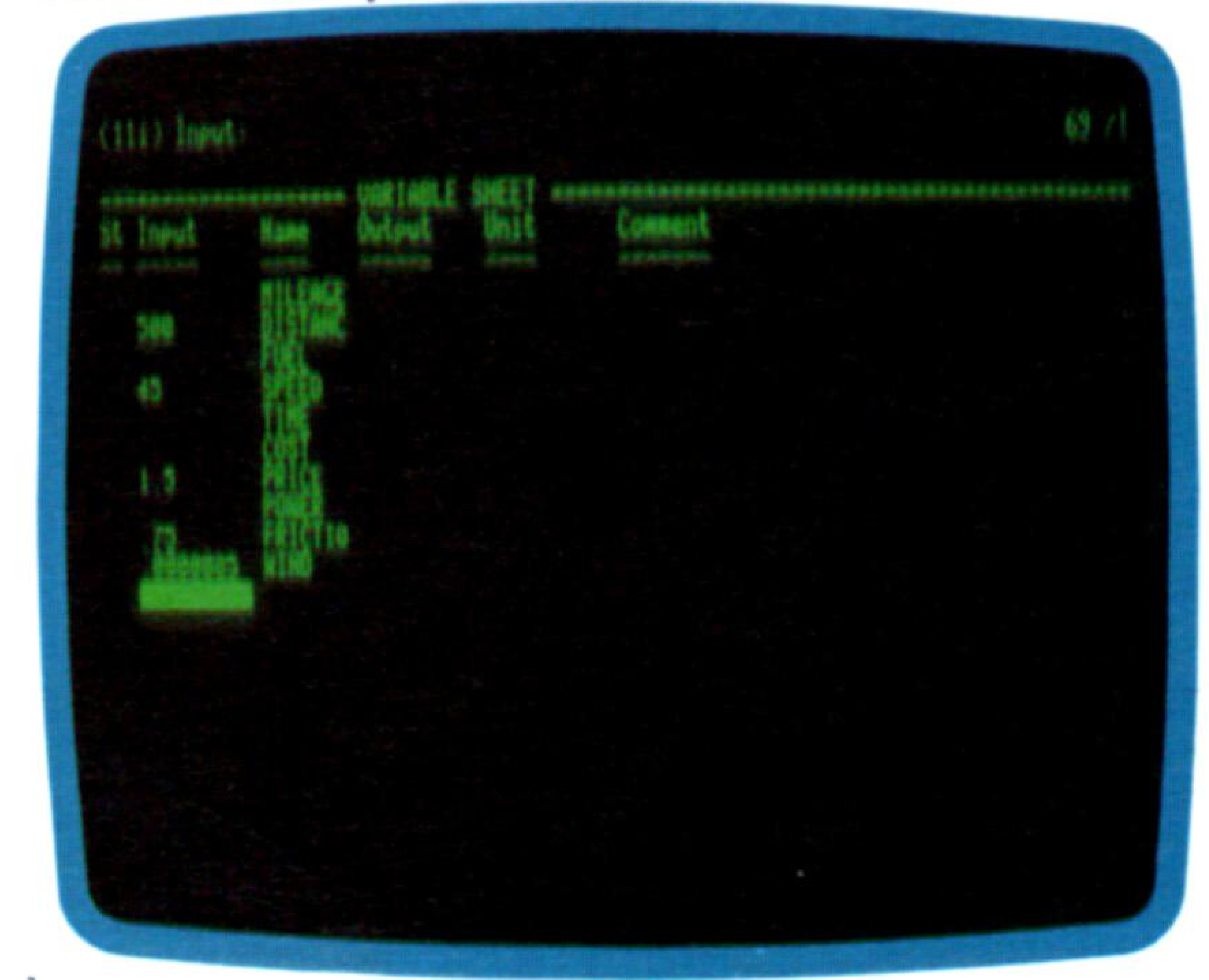


variables, la pantalla queda demasido pequeña para contener toda la información. Para ver visualizadas todas las variables, podemos hacer que la hoja Variable aparezca sola en una ventana. Esto lo conseguimos pulsando la tecla del punto y coma (;) para trasladar al cursor hasta la ventana de variables, y digitando luego W1. Ahora se pueden ver todas las variables y podemos comenzar a entrar valores para ellas.

Solución directa

El modelo se puede resolver directamente si al principio se le suministra información suficiente.

Entrada de valores para el Direct Solver



Por ejemplo, si entra los siguientes valores en la columna INPUT: 5000 a DISTANCIA, 45 a VELOCIDAD, 1.5 a PRECIO, .75 a FRICCION y .0000085 a VIENTO, y después pulsa ! para llevar a cabo los cálculos, *TK!*

Resultados del Direct Solver



visualizará el mensaje Direct Solver, y los valores desconocidos aparecerán en la columna OUTPUT.

Ello nos proporciona un claro análisis de toda la información que deseamos. Pero, ¿y si deseáramos utilizar este modelo para resolver un problema habiendo dado desde el comienzo menos información? Tomemos un cálculo en el cual tenemos un presupuesto máximo de 50 libras para gastar en gasolina en un viaje de 1 000 millas. Sabemos el precio de la gasolina (1.75 libras el galón), de modo que podemos determinar cuánto podemos gastar por milla. Sería más difícil, sin embargo, determinar a qué velocidad debemos viajar para completar el recorrido sin pasarnos del presupuesto.

Empezamos por poner a cero los valores que ya habíamos entrado. Esto lo hacemos digitando RVY (de *Reset Variables Yes*: poner variables a cero sí). Luego entramos la información que ya conocemos: 1000 para distancia, 50 para costo, y 1.75 libras para precio. Utilizamos un valor de 1/3 para la fricción interna (que *TK!* redondea a 0.333333) y 0.0000095 para la resistencia del viento. Pulse !

Modelo incompleto



para calcular, y aparecerán los valores correspondientes. Podrá observar que no se genera ningún valor para velocidad, tiempo y potencia, y la velocidad es el valor específico que necesitamos. Si pasamos la visualización de la hoja Variable a la hoja Rule, veremos que tres de nuestras ecuaciones

Ecuaciones no satisfechas



están sin satisfacer (lo que se indica mediante el * de la columna Status).

Solucionador iterativo

Dado que no podemos resolver el modelo utilizando el Direct Solver, debemos probar con el Iterative Solver (solucionador iterativo). Éste toma un valor inicial, entrado a modo de conjetura o hipótesis, y lo acomoda en la ecuación. Si el valor no es correcto, el *TK!Solver* emplea una serie de aproximaciones sucesivas para determinar con precisión el valor exacto.

Lo primero que debemos hacer es tomar el valor de gastos de viaje generado previamente y trasladarlo a la columna INPUT para suministrarle a *TK!* un valor extra con el cual comenzar. Esto lo hacemos digitando I en la columna Status junto a gastos de viaje en la hoja Variable. Luego estimamos un valor para velocidad (50, p. ej.); entre este número en la columna Input, digite G de *guess* (conjetura) en la columna Status y pulse ! para calcular. *TK!*

Valores iterados



visualiza Iterative Solver en la parte superior de la pantalla y un contador de cada aproximación. Al cuarto intento, *TK!* obtiene el valor correcto para velocidad, tiempo y potencia.

Según el programa, se necesita una velocidad promedio de poco más de 47 millas por horas para completar el viaje con el presupuesto fijado. Cuanto más próxima sea la conjetura al valor verdadero, más pronto hallará *TK!* la solución.

Orden cumplida

En este capítulo veremos cómo nuestro proyecto de programación analiza y obedece las instrucciones que le imparte el jugador

Banderas a cuadros

Las banderas (o *flags* o indicadores) son de uso común en aquellos programas que poseen una construcción modular. Dentro de un módulo se pueden comprobar las condiciones que implican una bifurcación en el control del programa, pero la bifurcación que pueda producirse a raíz del resultado de dicha condición se puede demorar hasta que se realice un retorno a la sección del programa principal. El establecimiento de una variable en un valor preestablecido cuando se efectúa la comprobación, nos permite que el valor de la variable se pueda comprobar más tarde dentro de la sección del programa principal. Las variables que se emplean de esta manera para señalar condiciones se denominan *flags* (o banderas o indicadores). El diagrama de flujo muestra el bucle principal del programa de *El bosque encantado*, tal como lo tenemos construido hasta el momento. La bandera F indica si una instrucción tiene un formato válido o no, y se establece durante la subrutina de "descomposición de instrucciones". La subrutina utilizada para identificar y ejecutar instrucciones normales emplea dos banderas: VF para indicar que la parte del verbo de la instrucción se ha reconocido correctamente. Si al ejecutar una instrucción el jugador se desplaza hasta un escenario nuevo, el hecho de que se ha efectuado un desplazamiento se indica mediante MF. Cuando se comprueba MF dentro del bucle principal del programa, un valor de 1 indica que el bucle ha de bifurcar hacia atrás para describir el nuevo escenario hasta el cual se ha desplazado el jugador

Ian McKinnell

Las aventuras por lo general se construyen de modo tal que el jugador pueda desplazarse de un escenario a otro, recogiendo o abandonando objetos a lo largo del camino. Para llevar a cabo estas tareas hemos empleado estas instrucciones:

AVANZAR	Para desplazarse por los escenarios
RECOGER (objeto)	Para coger un objeto
DEJAR (objeto)	Para deshacerse de un objeto
LISTAR	Para hacer una lista de objetos
MIRAR	Para volver a visualizar la descripción del escenario actual
FIN	Para dar por finalizado el juego

También se podría disponer de variantes de estas instrucciones, tales como IR en vez de AVANZAR, o TOMAR en vez de RECOGER. Parte de la diversión que ofrece un juego de aventuras es determinar cuáles son las palabras que aceptará el juego. Por ejemplo, un jugador podría probar con la instrucción NADAR estando en un escenario seco. Si el programa responde diciéndole al jugador que no puede nadar *aquí*, el jugador podría lógicamente suponer que existen escenarios en los cuales *sí* está permitido nadar. (También existe la posibilidad de que el programador quiera inducir al jugador a pensar que ¡efectivamente es así!)

La cantidad de instrucciones que acepta un juego varía a tenor de la complejidad del juego y de la cantidad de esfuerzo que ha invertido el programador para prever cualquier posible eventualidad. Al crear el programa es esencial que el diseñador se asegure de que el programa no se interrumpa si el jugador trata de entrar una instrucción que no se haya previsto. Todo lo que se requiere es una rutina de autoprotección que imprima "No comprendo", teniendo presente que el programa debe poseer la flexibilidad suficiente para que los jugadores puedan entrar instrucciones de diferentes formas. Por ejemplo, sería muy molesto que un programa aceptara la instrucción TOMAR LAMPARA y respondiera a la instrucción TOMAR LA LAMPARA con un "No comprendo". La incorporación de flexibilidad la analizaremos con mayor profundidad más adelante. Por el momento, hemos de considerar la clase de instrucciones que se podrán impartir durante el juego e idear una rutina que descomponga las mismas de modo tal que puedan ser interpretadas fácilmente.

Descomponer instrucciones

Independientemente de lo que signifique la instrucción, es muy probable que esté construida en modo *imperativo*, como, por ejemplo, AVANZAR SUR HACIA EL RIO o MATAR AL EXTRATERRESTRE. La ventaja de esta estructura de oración es que es fácil de descomponer: el verbo siempre ocupa el primer lugar de la oración, seguido por el objeto del verbo

y, en último lugar, puede haber una cualificación de la acción. La primera etapa del análisis de una instrucción consiste en separar el verbo del resto de la frase. Esta tarea se puede realizar fácilmente explorando la oración carácter a carácter, utilizando MID$, hasta hallar un espacio. La parte de la frase que se halla a la izquierda del espacio es el verbo, y puede ser asignado a la variable VB$. La parte de la frase que hay hacia la derecha puede ser asignada a una segunda variable, NN$. Esta subrutina se utiliza en *El bosque encantado* para descomponer la instrucción contenida en la variable IS$:

```
2500 REM **** S/R DESCOMPONER INSTRUCCION ****
2510 IF IS$="LISTAR" OR IS$="FIN" THEN VB$=IS$:F=1:RETURN
2515 IF IS$="MIRAR" THEN VB$=IS$:F=1:RETURN
2520 F=0
2530 LS=LEN(IS$)
2540 FOR C=1 TO LS
2550 A$=MID$(IS$,C,1)
2560 IF A$<>" " THEN 2590
2570 VB$=LEFT$(IS$,C-1):F=1
2580 NN$=RIGHT$(IS$,LS-C):C=LS
2590 NEXT C
2600 :
2610 IF F=1 THEN RETURN
2620 PRINT:PRINT"NECESITO AL MENOS DOS PALABRAS"
2630 RETURN
```

La rutina, antes de intentar descomponer la oración, verifica primero que la instrucción no sea ninguna de las tres instrucciones posibles compuestas por una sola palabra, es decir, LISTAR, MIRAR o FIN. Si se trata de una instrucción de una sola palabra, toda ella es asignada a VB$ y se sale de la rutina. Si la instrucción no es ninguna de estas tres, entonces la rutina se introduce en un bucle FOR...NEXT y comienza a explorar en busca del primer espacio. En este bucle se utilizan dos técnicas que merecen una mención especial. Ambas están relacionadas con el hecho de que efectuar un salto condicional para salir fuera de un bucle FOR...NEXT sin pasar por la sentencia NEXT es un estilo de programación sumamente malo. En cambio, para señalar que se ha satisfecho alguna condición (en este caso, que se ha encontrado un espacio) se establece un flag o indicador, F, en uno. En segundo lugar, cuando se ha hallado el primer espacio, es una pérdida de tiempo seguir explorando el resto de la frase.

El bucle se puede terminar limpiamente en este punto estableciendo el contador del bucle, C, en su límite superior, LC. Por consiguiente, cuando el programa vuelva a llegar a NEXT, pasará a la instrucción siguiente, en vez de saltar otra vez hacia atrás hasta la sentencia FOR. Una vez terminado correctamente el bucle, se puede comprobar el estado del indicador F. Un valor de uno en el indicador significa que la frase está compuesta por más de una palabra, y todo cuanto resta por hacer en esta etapa es retornar al bucle principal. Si el indicador no es uno, entonces la instrucción sólo posee una palabra y no es ninguna de las instrucciones de una sola palabra comprobadas previamente. En este caso, antes de retornar en busca de otra instrucción, se imprime un mensaje que señala que se requieren dos palabras.

Instrucciones normales

En la mayor parte del programa el jugador sencillamente se trasladará de un escenario a otro y recogerá o abandonará los objetos que pueda haber encontrado. Por consiguiente, para la mayoría de los escenarios, las instrucciones AVANZAR, RECOGER, DEJAR, LISTAR, MIRAR, FIN (y sus variantes) son suficientes para permitir que el jugador realice esto. Sólo en circunstancias inusuales el jugador deseará utilizar otras instrucciones más especializadas. Por ejemplo, no tiene mayor sentido emplear la instrucción MATAR si no hay nada que poder matar. No obstante, es posible idear una estructura de programa en la cual, en la mayor parte de las ocasiones, sólo se comprueben las seis instrucciones relativas a movimiento y objetos. Cuando el jugador entra en un nuevo escenario, el programa puede verificar si se trata de uno que, por algún motivo, se haya señalado como "especial". Si fuera éste el caso, entonces cualquier nuevo requisito para instrucciones se puede tratar mediante una subrutina de instrucciones específica para ese escenario en particular. Por lo tanto, el bucle principal de llamada de nuestro programa debería hacer lo siguiente:

1) Describir el escenario y listar las salidas.
2) Determinar si éste es "especial".
3) Solicitar una instrucción y, si aquél no es especial, explorar la lista de instrucciones normales.

En el bucle principal ha de haber, asimismo, una facilidad para distinguir entre una instrucción que produce un desplazamiento a un escenario nuevo y otra que no lo produzca. En el primer caso, el bucle necesita volver atrás hasta el comienzo del bucle, para describir el nuevo escenario y decidir si es o no especial. En el segundo caso, sólo es necesario saltar hacia atrás para solicitar una nueva instrucción. La forma más simple de implementar esto consiste en utilizar un "indicador de movimiento", MF, que normalmente está establecido en cero. Si una instrucción implica movimiento, entonces este indicador se establece en uno. El estado de MF se puede comprobar al final del bucle principal, efectuando el salto adecuado. Agréguele a *El bosque encantado* las siguientes líneas:

```
270 GOSUB2500:REM DESCOMPONER INSTRUCCION
275 IF F=0 THEN 260:REM INSTRUCCION NO VALIDA
280 GOSUB3000:REM INSTRUCCIONES NORMALES
290 IF VF=0 THENPRINT:PRINT"NO COMPRENDO"
300 IF MF=1 THEN 240:REM NUEVO ESCENARIO
310 IF MF=0 THEN 260: REM NUEVA INSTRUCCION

3000 REM **** S/R INSTRUCCIONES NORMALES ****
3010 VF=0: REM INDICADOR VERBO
3020 IF VB$="AVANZAR" OR VB$="IR" THEN VF=1:GOSUB3500
3030 IF VB$="RECOGER" OR VB$="TOMAR"THEN VF=1:GOSUB3700
3040 IF VB$="DEJAR" OR VB$="PONER"THEN VF=1:GOSUB3900
3050 IF VB$="LISTAR" OR VB$="INVENTARIO" THEN VF=1:GOSUB4100
3055 IF VB$="MIRAR" THEN VF=1:MF=1:RETURN
3060 IF VB$="FIN" OR VB$="TERMINAR" THEN VF=1:GOSUB4170
3070 RETURN
```

En la primera rutina se utiliza otro indicador, VF, para indicar si el verbo se ha entendido y obedecido o no. VF se establece en uno sólo cuando se ha aislado al verbo. En el bucle principal podemos insertar una sentencia "No comprendo" de autoprotección, comprobando el estado de VF. Si VF permanece en cero, entonces la rutina de análisis no ha reconocido el verbo y se visualiza la sentencia.

En el próximo capítulo del proyecto nos ocuparemos de las subrutinas para recoger, abandonar y listar objetos. De momento, podemos agregarle a nuestro grupo de instrucciones normales una corta subrutina de instrucción FIN:

```
4170 REM **** S/R FIN DEL JUEGO ****
4180 PRINT:PRINT"ESTAS SEGURO (S/N)  ?"
4190 GET A$:IF A$<> "S" AND A$<>"N" THEN 4190
4200 IF A$="N" THEN RETURN
4210 END
```

La instrucción MIRAR también es directa. Para volver a describir el escenario actual, simplemente hemos de establecer el "indicador de movimiento", MF, en uno, y retornar al bucle principal del progra-

ma. El establecimiento de MF hará que el programa salte hacia atrás hasta el principio, llamando por tanto a las rutinas que describen un escenario y sus salidas. Dado que el valor de la variable de escenario, P, no sufre ninguna alteración a causa de la instrucción MIRAR, se describirá el mismo escenario. Esta instrucción es útil si, después de que el jugador haya llevado a cabo una serie de acciones, la descripción original del escenario actual se hubiera desplazado fuera de la pantalla.

Añadir flexibilidad

Al impartir instrucciones de movimiento, el jugador puede digitar diferentes formas de la misma instrucción. Por ejemplo, AVANZAR NORTE, IR NORTE y AVANZAR HACIA EL NORTE están solicitando lo mismo. A pesar de que no es de vital importancia que un programa de juego de aventuras reconozca todas estas formas, el hecho de que diversos formatos de instrucción sean legales contribuye a dotar al juego de un mayor interés. Las tres instrucciones de movimiento que acabamos de ofrecer poseen una estructura común: todas empiezan con un verbo de movimiento y la dirección requerida es una palabra discreta. Se puede, por consiguiente, diseñar una rutina que busque la dirección en la parte de la frase que viene después del verbo. La rutina explora esta zona de la oración en busca de espacios, aislando cada palabra y comparándola con las cuatro palabras de dirección buscadas hasta hallar un emparejamiento.

```
3630 REM **** S/R BUSCAR DIRECCION ****
3640 NN$=NN$+" ":LN=LEN(NN$):C=1
3645 FOR I=1 TO LN
3650 IF MID$(NN$,I,1)<>" " THEN NEXT I:RETURN
3655 W$=MID$(NN$,C,I-C):C=I+1
3660 IF W$="NORTE" OR W$="ESTE" THEN NN$=W$:I=LN
3665 IF W$="SUR" OR W$="OESTE" THEN NN$=W$:I=LN
3670 NEXT I
3675 RETURN
```

En el capítulo anterior del proyecto desarrollamos una rutina de movimiento. Para añadir esta nueva rutina a la de movimiento sólo hemos de agregar la siguiente línea:

```
3505 GOSUB 3630:REM BUSCAR DIRECCION
```

Vale la pena destacar que esta rutina no obedecerá instrucciones tales como AVANZAR EN UNA DIRECCION NORTEÑA, puesto que la rutina no puede aislar la palabra de dirección. Sería posible diseñar una rutina que trabajara en función del principio de explorar grupos de cuatro y cinco letras, comparando cada grupo con las cuatro palabras de dirección posibles. Sin embargo, tal rutina tomaría mucho tiempo de ejecución. Por otra parte, nuestro programa sí aceptará AVANZAR NORTEÑA, dado que la rutina de movimiento finalmente utiliza la primera letra de la segunda parte de la oración, NN$. En este caso, la N de NORTEÑA se aceptaría como si se tratara de la N de NORTE.

Listados Digitaya

```
1220 GOSUB1700:REM ANALIZAR INSTRUCCIONES
1225 IF F=0 THEN 1210:REM INSTRUCCION NO VALIDA
1230 GOSUB 1900:REM INSTRUCCIONES NORMALES
1240 IF VF=0 THEN PRINT"NO COMPRENDO"
1250 IF MF=1 THEN 1160:REM NUEVA POSICION
1260 IF MF=0 THEN 1210:REM NUEVA INSTRUCCION
1700 REM **** S/R ANALIZAR INSTRUCCION ****
1705 F=0:REM BANDERA CERO
1710 IF IS$="FIN" OR IS$="LISTAR" THEN VB$=IS$:F=
     1:RETURN
1720 IF IS$="MIRAR" THEN VB$=IS$:F=1:RETURN
1730 :
1740 REM ** DESCOMPONER INSTRUCCION **
1750 VB$="":NN$="":REM VERBO Y SUSTANTIVO CERO
1770 LS=LEN(IS$)
1780 FOR C=1 TO LS
1790 A$=MID$(IS$,C,1)
1800 IF A$=" " THEN VB$=LEFT$(IS$,C-1):NN$=RIGHT$
     (IS$,LS-C):F=1:C=LS
1810 NEXT
1830 IF F=0 THEN PRINT:PRINT"NECESITO AL MENOS DOS
     PALABRAS"
1840 RETURN
1850 :
1900 REM **** S/R ACCIONES NORMALES ****
1910 VF=0
1920 PRINT
1930 IF VB$="AVANZAR" OR VB$="IR" THEN VF=1:GOSUB2000
1940 IF VB$="RECOGER" OR VB$="TOMAR" THEN VF=
     1:GOSUB2140
1950 IF VB$="DEJAR" OR VB$="PONER" THEN VF=1:GOSUB2360
1960 IF VB$="LISTAR" OR VB$="INVENTARIO" THEN
     VF=1:GOSUB2540
1965 IF VB$="MIRAR" THEN VF=1:MF=1:RETURN
1970 IF VB$="FIN" OR VB$="TERMINAR" THEN VF=1:GOSUB2610
1980 RETURN

2015 GOSUB8600:REM BUSCAR DIRECCION
2610 REM **** S/R FIN DEL JUEGO ****
2620 PRINT:PRINT"ESTAS SEGURO (S/N) ?"
2630 GETA$:IFA$<>"S" AND A$<>"N"THEN2630
2640 IFA$="N"THEN RETURN
2650 END

8600 REM **** S/R BUSCAR DIRECCION ****
8610 NN$=NN$+" ":LN=LEN(NN$):C=1
8620 FOR I=1 TO LN
8630 IF MID$(NN$,I,1)<>" " THEN NEXT I:RETURN
8640 W$=MID$(NN$,C,I-C):C=I+1
8650 IF W$="NORTE" OR W$="ESTE" THEN NN$=W$:I=LN
8660 IF W$="SUR" OR W$="OESTE" THEN NN$=W$:I=LN
8670 NEXT I
8680 RETURN
```

Complementos al BASIC

Spectrum:
En ambos programas, utilice I$ por IS$, B$ por VB$ y R$ por NN$.
En *Digitaya*, reemplace las siguientes líneas:

```
1790 LET A$=I$(C TO C)
1800 IF A$="" THEN LET B$=I$(TO C-1):LET
     R$=I$ (LEN(I$)-LS+C+1 TO): LET F=
     1:LET C=LS
2630 LET A$=INKEY$:IF A$<>"S"
     AND A$<>"N" THEN 2630
8630 IF R$(I TO I)<>"" THEN NEXT I:RETURN
8640 LET W$=R$(C TO I-1):LET C=I+1
```

En *El bosque encantado* sustituya estas líneas:

```
2550 LET A$=I$(C TO C)
2570 LET B$=I$(TO C-1):LET F=1
2580 LET R$=I$(LEN(I$)-LS+C+1 TO):LET C=LS
3650 IF R$(I TO I)<>"" THEN NEXT I:RETURN
3655 LET W$=R$(C TO I-1):LET C=I+1
4190 LET A$=INKEY$:IF A$<>"S" AND
     A$<>"N" THEN GOTO 4190
```

BBC Micro:
En *Digitaya* reemplace esta línea:

```
2630 REPEAT:A$=GET$:UNTIL A$="S"
     OR A$="N"
```

y ésta en *El bosque encantado*:

```
4190 REPEAT:A$=GET$:UNTIL A$="S"
     OR A$="N"
```

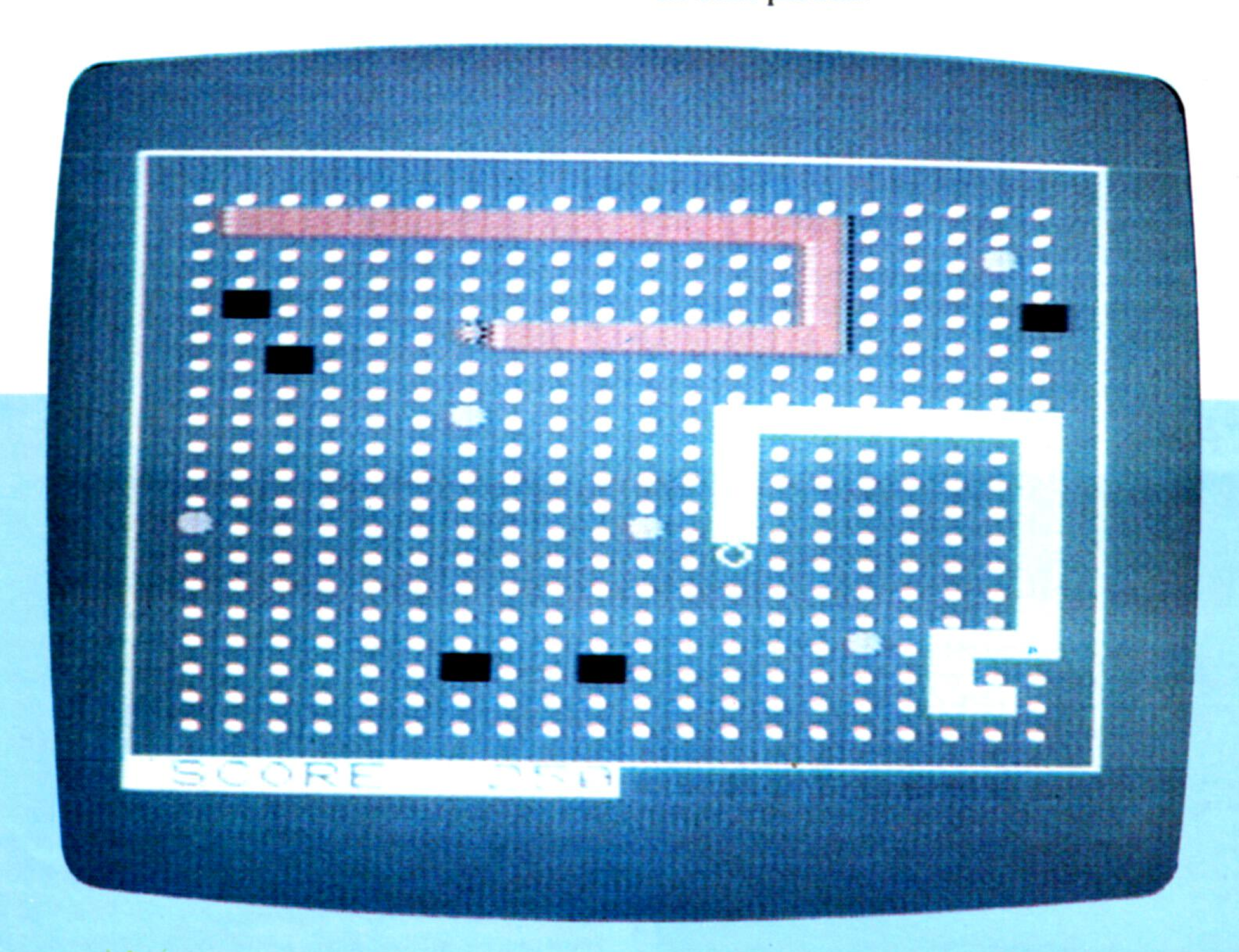
La trampa

¡Atención a la corriente eléctrica!: ella constituye "la trampa". El listado de este interesante juego ha sido escrito para el Vic-20

En este juego, usted se desplaza dentro de un rectángulo dejando una estela tras de sí. El ordenador controla un robot que, al trasladarse de lugar, deja asimismo un trazo a sus espaldas. Si usted toca los límites del rectángulo o si corta un trazo, se electrocutará.

Los puntos se cuentan de la manera siguiente: Si el robot se detiene (el círculo verde): 200 puntos; si usted pasa sobre un punto malva: 50 puntos; si usted pasa sobre un punto blanco: 10 puntos.

La velocidad en la ejecución del juego aumenta en cada partida.




```
10 RESTORE
13 GOSUB3000:GOSUB2000
15 X=RND(-TI):DEFFNR(X)=(PEEK(7663+X)=0)
20 D(0)=1:D(1)=22:D(2)=-1:D(3)=-22
30 L=150+(KJ$="J")*50
40 PP=46:PC=437:DP=1:DC=2:SC=7680:CL=38400:POKE36878,15
45 GOSUB1000
50 IFKJ$="J"THEN62
55 GETA$:POKE198,0
60 ND=22*((A$="P")-(A$="."))+(A$="L")-(A$=":"):IFND<>0THENDP=ND
61 GOTO70
62 SYS7600:X1=0:FORX=0TO3:X1=X1-FNR(X):NEXT:IFX1>1THEN70
65 ND=22*(FNR(0)-FNR(1)-FNR(2)-FNR(3):IFND<>0THENDP=ND
70 NP=PP+DP:X1=PEEK(SC+NP):IFX1<>46THEN700
75 POKE36874,128:S=S+10
80 POKESC+PP,160:POKECL+PP,2:POKESC+NP,42:POKECL+NP,2:PP=NP
85 PRINT PTS "S" ";:POKE36874,0:POKE36876,0
90 NC=PC+D(DC):IFPEEK(SC+NC)<>460RRND(1)<,15THEN200
100 POKESC+PC,160:POKECL+PC,5:POKESC+NC,15:POKECL+NC,5:PC=NC
110 FORX=0TOL:NEXT:GOTO50
200 FORX=1TO19:DC=INT(RND(1)*4):IFPEEK(SC+PC+D(DC))=46THENNC=PC+D(DC):GOTO100
210 NEXT
220 S=S+200:L=L/2
230 FORX=1TO4:FORY=128TO160:POKE36875,Y:NEXT:NEXT:POKE36875,0:POKE198,0:GOTO40
700 IFX1=81THENPOKE36876,200:S=S+50A:GOTO80
800 POKE36877,250:FORX=1TO2000:NEXT:POKE36877,0
810 PRINT"          FINALIZADO          ":FORX=1TO5000:NEXT:POKE198,0:S=0:GO
    TO10
1000 PRINT"                    ";:FORX=1TO20:PRINT"| ....................  |"
     ;:NEXT
1010 PRINT"                    ";
1020 FORX=1TO5:X1=INT(RND(1)*20)+1+(INT(RND(1)*28)+1)*22
1030 POKE7680+X1,160:POKE38400+X1,0:NEXT
1040 FORX=1AT5:X1=INT(RND(1)*20)+1+(INT(RND(1)*20)+1)*22
1050 POKE7680+X1,81:POKE38400+X1,4:NEXT:POKESC+46,46:POKESC+437,46
1060 POKECL+46,1:POKECL+437,1:RETURN
2000 POKE36879,25
2010 PRINT"
2020 PRINT"TRAMPA "DESPLACESE UTILIZANDO LA PALANCA DE MANDO O"
2030 PRINT"                 P   L  :   ,  "
2040 PRINT"PTS-":PRINT".-10 PTS":PRINT-50 PTS":PRINT"ROBOT DETENIDO-200 P
     TS"
2960 PRINT"PULSE UNA TECLA
2970 GETA$:IFA$=""GOTO2970
2980 POKE36879,110:IFKJ$<>""THENRETURN
2990 PRINT"TECLADO O PALANCA DE MANDO?
2991 GETCM$:IFCM$>>"C"ANDCM$<>"M"THEN2991
2992 RETURN
3000 POKE51,176:POKE52,29:POKE55,176:POKE56,29
3010 FORX=7600TO7661:READA:POKEX,A:NEXT
3020 RETURN
3030 DATA169,0,141,19,145,169,127,141,34,145,173,17,145,141,238,29,169,4,45,238,
     29
3040 DATA141,239,29,169,8,45,238,29,141,240,29,169,16,45,238,29,141,241,29,169,32
3050 DATA45,238,29,141,243,29,173,32,145,41,128,141,242,29,169,255,141,34,145,96
```

Toque de distinción

He aquí un dispositivo para diseñar gráficos que se destaca especialmente porque puede ser utilizado por la mayoría de ordenadores personales

Todos los ordenadores de mayor venta en la actualidad ofrecen visualizaciones de gráficos en alta resolución. Sin embargo, a menos que se disponga de software para gráficos ya escrito, la creación de tales visualizaciones exige mucho tiempo y esfuerzo y muchas de las características no se utilizan al completo. Un programa para dibujar "bocetos" resulta insuficiente, porque a menudo el usuario desea copiar en el ordenador una imagen ya existente en lugar de simplemente dibujar a mano alzada.

Con esta finalidad se han comercializado varios digitalizadores, pero casi todos se han diseñado básicamente para ser utilizados con máquinas específicas, como el BBC Micro o el ZX Spectrum. La tablilla para gráficos Touchmaster está diseñada para trabajar con una amplia gama de máquinas personales (algunas de las cuales requerirán un cable o una interface apropiada). Este dispositivo se está promocionando, asimismo, como teclado de recambio, pero la simplicidad de su diseño implica que dicha utilización se ve limitada a la selección de opciones de menú o al control de juegos. Para la entrada de datos, así como para la carga del propio software del Touchmaster, sigue siendo necesario un teclado de ordenador.

El teclado del Touchmaster
La hoja que se proporciona con el software se encaja simplemente en la superficie de dibujo. La instrucción se ejecuta en la pantalla al hacer presión sobre la instrucción apropiada en el lado derecho de la cubierta y mover el lápiz, el punzón o el dedo hasta la superficie de dibujo



Ian McKinnell

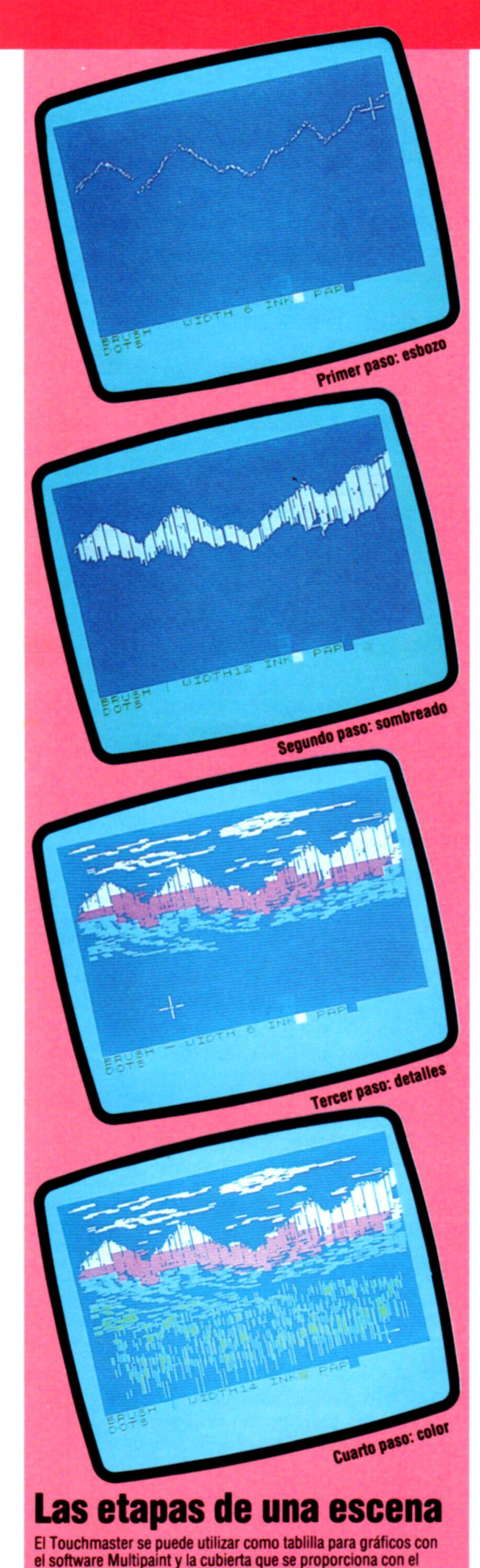
El Touchmaster viene instalado en una carcasa plástica de color gris que mide 350×330×35 mm. La parte posterior de la misma está ligeramente elevada, formando un ángulo adecuado para dibujar. Se suministra con un transformador y un LED rojo que indica si el dispositivo está conectado o no. Para que la tablilla se pueda utilizar con una amplia gama de máquinas personales, en el panel posterior hay instalados conectores para interfaces tanto en serie como en paralelo, junto con un conector (que no se menciona en los manuales) para un interruptor de pie. En realidad, los manuales no son muy adecuados: el de hardware ofrece instrucciones para la conexión de la tablilla y proporciona algunos programas sencillos en BASIC para leer coordenadas, pero carece de los suficientes detalles.

La tablilla se basa en la tecnología de membrana desarrollada en los teclados del ZX81 y del Spectrum, y ofrece una resolución de 256×256 pixels. La capa superior está separada de la película sensitiva inferior por un tejido aislante, y la presión efectuada sobre la capa superior la obliga a hacer contacto con la película. La tablilla contiene un microprocesador que explora la película sensitiva en una dirección, mientras explora al mismo tiempo la capa inferior en otra dirección, y la coordenada del "punto de contacto" se envía entonces a través de interfaces tanto en serie como en paralelo. La interface en serie se utiliza para conectar la tablilla al BBC Micro, mientras que la interface en paralelo es necesaria para emplearla con el Commodore 64, el Vic-20, el Spectrum y el Dragon. La resolución del Touchmaster es inferior a la que proporcionan muchas visualizaciones en pantalla de alta resolución, de modo que los usuarios del BBC Micro, por ejemplo, no podrán resolver un pixel individual en Mode 0.

Programa Multipaint

Con el Touchmaster se proporciona un programa de dibujo llamado *Multipaint*. Éste ofrece una demostración de las facilidades disponibles, pero no representa una gran ayuda para los gráficos. Una plantilla plástica proporciona un menú de las facilidades, con la opción elegida visualizada en una ventana de "estado" en la parte inferior de la pantalla. Se pueden utilizar cinco tipos distintos de pincel; el ancho de cada uno puede oscilar entre 2 y 32 pixels, en pasos de dos pixels. La ventana muestra,





Las etapas de una escena

El Touchmaster se puede utilizar como tablilla para gráficos con el software Multipaint y la cubierta que se proporciona con el producto. En las fotografías vemos las etapas de una escena, desde el bosquejo hasta la imagen coloreada y sombreada

asimismo, la modalidad en curso de dibujo (Dots, Points o Freehand: pixels, puntos o mano alzada) y los colores seleccionados para primer plano y fondo. Éstos se pueden cambiar pulsando la opción requerida del menú hasta que aparezca en la ventana de estado.

Una vez se han seleccionado los colores y los tipos de pincel, hay otras opciones disponibles para la creación de cajas, círculos, polígonos y líneas "elásticas". Con el paquete se proporciona un punzón, pero también se puede utilizar la presión de los dedos. El Touchmaster, con la gran superficie de su tablilla, no se ve afectado por las mismas restricciones del Koala-pad; la presión de un dedo se puede traducir a una coordenada exacta y no a una mera aproximación, ¡de modo que el dibujo electrónico a dedo es realmente posible!

Lamentablemente, el Touchmaster no ofrece más que facilidades rudimentarias. Hay una opción FILL (rellenar) marcada en la plantilla y documentada en los manuales, pero (al menos en el Spectrum) la misma no parece funcionar de la forma esperada. Tampoco hay una facilidad para ampliación ni para edición, lo que significa que los colores no se pueden cambiar. En el Spectrum, con el cual suele ser más fácil dibujar en blanco y negro antes de añadir color, esto constituye una evidente desventaja.

Como elemento de hardware, la tablilla Touchmaster está en una posición mucho mejor frente a rivales como el Grafpad y el Koala-pad. Es de construcción sólida y ofrece una superficie de dibujo de tamaño A4 que se puede conectar a la mayoría de los ordenadores personales más populares. Una ventaja significativa es que si en el futuro usted decide ampliar su máquina o directamente cambiarla por otra, todo lo que se requiere es una nueva interface; además, por supuesto, del software adecuado.

Resulta decepcionante que la documentación y el software suministrados sean tan pobres en comparación con el estándar de la tablilla. El Touchmaster está dando origen a una gama de software diseñado específicamente para usar con esta tablilla, si bien la verdadera prueba de su éxito se producirá cuando las casas de software independientes decidan apoyarla.

TOUCHMASTER MSX

DIMENSIONES

350×330×35 mm

INTERFACES

Se proporcionan puertas en serie y en paralelo para utilizar con una vasta gama de ordenadores personales o con el propio teclado del Touchmaster

DOCUMENTACION

Si bien los manuales contienen la información necesaria para utilizar el Touchmaster, hubiera sido de agradecer una información más detallada

VENTAJAS

La amplia gama de interfaces, junto con los diferentes paquetes de software, hacen del Touchmaster un periférico sumamente adaptable

DESVENTAJAS

La calidad del software queda en una posición muy precaria si se la compara con la de dispositivos similares; los manuales son limitados



De colores

Existen diversas utilidades y programas para juegos, de entretenimiento y educativos a la venta para el Touchmaster en las principales tiendas especializadas. Cada paquete incluye el software apropiado (para una gama de micros), instrucciones e imaginativas cubiertas, algunas de las cuales se pueden apreciar aquí

¿Quién lo hizo?

Mostramos la preparación de una base de datos sencilla en un ejemplo que propone la investigación de un crimen

En una pequeña localidad de los Andes se ha perpetrado un asesinato. Zacarías ha sido atacado con un hacha y muerto. Sabemos que tanto Mateo como José poseen hachas, que Jaime y Esteban poseen escopetas y que la prima Juana tiene un cuchillo. Mateo y Jaime tenían sangre en las manos cuando el comisario local los interrogó.

Nuestra base de datos en LOGO sobre este crimen consistirá en una lista de *hechos*, cada uno de los cuales consistirá en una *relación*, junto con uno o más sustantivos. Al representarlo en LOGO, un hecho es [POSEE MATEO HACHA] o, en correcto castellano, "Mateo posee un hacha". Para representar el hecho de que Jaime tenía sangre en las manos utilizamos [ENSANGRENTADO JAIME].

Comenzamos nuestra investigación con una base de datos vacía:

```
TO PREPARACION
  MAKE "BASEDATOS []
END
```

Luego vamos añadiendo hechos a nuestra base de datos a medida que los vamos descubriendo (siempre y cuando no estuvieran ya en ella). Por ejemplo, entraríamos AGREGAR [POSEE JUANA CUCHILLO] empleando este procedimiento:

```
TO AGREGAR :HECHO
  IF NOT MEMBER? :HECHO :BASEDATOS THEN
  MAKE "BASEDATOS FPUT :HECHO
  :BASEDATOS
END
```

La base de datos finalmente estará llena con:

```
[[ENSANGRENTADO MATEO] [ENSANGRENTADO
JAIME] [ASESINADO
  ZACARIAS HACHA] [POSEE MATEO HACHA]
[POSEE JOSE HACHA] [POSEE JAIME ESCOPETA]
[POSEE ESTEBAN ESCOPETA] [POSEE JUANA
  CUCHILLO]]
```

Para imprimir la base de datos utilice el procedimiento MOSTRAR. El mismo puede ir seguido ya sea de "TODO, en cuyo caso se imprimirá toda la base de datos, o bien del nombre de una relación, en cuyo caso sólo se imprimirán los hechos de esa relación. Por consiguiente, MOSTRAR "POSEE nos mostrará quién posee qué.

```
TO MOSTRAR :S
  IF :S="TODO THEN LISTAR.TODO :BASEDATOS
  LISTAR.REL :S :BASEDATOS
END

TO LISTAR.TODO :LISTA
  IF EMPTY? :LISTA THEN STOP
  PRINT FIRST :LISTA
  LISTAR.TODO BUTFIRST :LISTA

END
TO LISTAR.REL :S :LISTA
  IF EMPTY? :LISTA THEN STOP
  IF :S=FIRST FIRST :LISTA THEN PRINT FIRST
    :LISTA
  LISTAR.REL :S BUTFIRST :LISTA
END
```

Ahora debemos pensar en formas de interrogar a la base de datos. La forma de interrogación más sencilla consiste en comprobar si se sabe que un hecho es verdadero. Esto lo hacemos mediante un procedimiento llamado PREG, que comprueba si un hecho está incluido en la base de datos. Por ejemplo, PREG [POSEE JUANA CUCHILLO] daría como respuesta SI.

```
TO PREG :HECHO
  IF MEMBER? :HECHO :BASEDATOS PRINT "SI
    ELSE PRINT "NO
END
```

Para nuestra investigación sería mucho más útil que pudiéramos formular preguntas como "¿Quién posee un hacha?". Esto lo manejaremos empleando "variables". Se dará por sentado que toda palabra cuyo primer carácter sea un ? es una variable. Entonces podemos parafrasear la pregunta así:

```
CUAL [POSEE ?ALGUIEN HACHA]
```

La respuesta a esto sería una lista de todos los valores posibles de la variable ?ALGUIEN que sean coherentes con la información de la base de datos.

```
[?ALGUIEN MATEO]
[?ALGUIEN JOSE]
NO (MAS) RESPUESTAS
```

Podemos tener múltiples variables. Por ejemplo:

```
CUAL [MATO ?HOMBRE ?UTENSILIO]
```

dará la respuesta:

```
[?HOMBRE ZACARIAS] [?UTENSILIO HACHA]
NO (MAS) RESPUESTAS
```

Consideremos individualmente los procedimientos que hacen viable este análisis de la base de datos. CUAL le pasa la tarea a HALLAR, indicando BASEDATOS como la fuente de hechos.

```
TO CUAL :INTERROGACION
  HALLAR :INTERROGACION :BASEDATOS
  PRINT [NO (MAS) RESPUESTAS]
END
```

HALLAR crea dos variables globales, VARS y RESP: VARS se utiliza para retener cada posible conjunto de valores de las variables de la pregunta, y éstos se reúnen en la lista RESP.

```
TO HALLAR :INTERROGANTES :DATOS
  MAKE "VARS []
  MAKE "RESP []
  COMPARAR :INTERROGACION :DATOS
  PRINTL :RESP
END
```



COMPARAR examina cada uno de los hechos de la base de datos de uno en uno. De haber un emparejamiento, el nuevo conjunto de valores de VARS se agrega a RESP antes de volver a colocar a VARS en la lista vacía. COMPARAR sigue entonces trabajando en la base de datos para ver si existe alguna otra posible pareja.

```
TO COMPARAR :INTERROGACION :DATOS
   IF EMPTY? :DATOS THEN STOP
   IF PAREJA? :INTERROGACION FIRST :DATOS
   THEN MAKE "RESP FPUT :VARS :RESP
   MAKE "VARS []
   COMPARAR :INTERROGACION BUTFIRST :DATOS
END
```

Para apreciar lo que hace PAREJA? considere el caso en el cual las entradas sean [POSEE ?ALGUIEN HACHA] y [POSEE JOSE HACHA], en respuesta a las cuales PAREJA? producirá TRUE y establecerá VARS en [?ALGUIEN JOSE]. Si las entradas fueran [POSEE ?ALGUIEN HACHA] y [MATO ZACARIAS HACHA], entonces PAREJA? produciría FALSE.

Las verdaderas dificultades surgen, sin embargo, cuando hay implicada más de una variable. Se emplea VALOR? para verificar si la variable ya se le ha asignado un valor para ese hecho en la base de datos.

Aquí hemos utilizado una notación alternativa para las condiciones del LOGO. TEST evalúa una condición. Si el resultado es verdadero, se llevan a cabo las acciones que van después de IFTRUE; de lo contrario, se efectúan las acciones que siguen a IFFALSE.

```
TO PAREJA? :INTERROGACION :HECHO
   IF ALLOF EMPTY? :INTERROGACION EMPTY?
      :HECHO THEN OUTPUT "TRUE
   TEST FIRST FIRST :INTERROGACION="?
   IFTRUE IF NOT VALOR? FIRST :INTERROGACION
      FIRST :HECHO :VARS THEN OUTPUT "FALSE
   IFFALSE IF NOT (FIRST   :INTERROGACION=
      FIRST :HECHO) THEN OUTPUT "FALSE
      OUTPUT PAREJA? BUTFIRST
      :INTERROGACION
   BUTFIRST :HECHO
END
```

Para ver cómo funciona VALOR?, consideremos en primer lugar el caso en el que las entradas sean ?UTENSILIO, HACHA y [?HOMBRE ZACARIAS]. VALOR? intenta determinar si la variable ?UTENSILIO podría tener el valor HACHA. Existen tres posibilidades: ?UTENSILIO ya posee un valor, que no es HACHA, y VALOR? produce FALSO; ?UTENSILIO ya posee el valor HACHA, y VALOR? produce VERDADERO; o ?UTENSILIO no posee ningún valor, de modo que se le da el valor HACHA, y se agrega esta información a VARS y se produce VERDADERO.

```
TO VALOR? :NOMBRE :VALOR :VLISTA
   IF EMPTY? :VLISTA THEN MAKE "VARS LPUT
      LIST :NOMBRE :VALOR :VARS OUTPUT
      "TRUE
   TEST :NOMBRE=FIR: T FIRST :VLISTA
   IFTRUE IF :VALOR=LAST FIRST :VLISTA THEN
      OUTPUT "TRUE ELSE OUTPUT "FALSE
   OUTPUT VALOR? :NOMBRE :VALOR BUTFIRST
      :VLISTA
END
```

PRINTL hace que los componentes de RESP se impriman uno debajo del otro.

```
TO PRINTL :LISTA
   IF EMPTY? :LISTA STOP
   PRINT FIRST :LISTA
   PRINTL BUTFIRST :LISTA
END
```

Interrogatorios más complejos

Nuestra investigación no llegará muy lejos, sin embargo, a menos que podamos formular preguntas más complejas, tales como "¿Con qué utensilio fue asesinado Zacarías y quién posee un utensilio como ése?". En LOGO, esto se lee:

```
CUAL [[MATO ZACARIAS ?UTENSILIO]
   [POSEE ?SOSPECHOSO ?UTENSILIO]]
```

Ahora CUAL toma como entrada una lista de preguntas, y los valores hallados serán aquellos que hagan que todas ellas resulten verdaderas. Entonces, si se desea formular una única pregunta con esta nueva forma de CUAL, la sintaxis que utilizamos es:

```
CUAL [[POSEE ?ALGUN CUCHILLO]]
```

En estos procedimientos sólo es necesario introducir unas ligeras modificaciones:

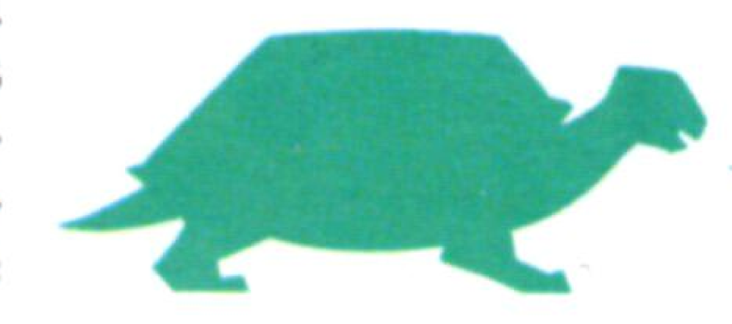
```
TO CUAL :INTERROGANTES
   HALLAR :INTERROGANTES :BASEDATOS
   PRINT [NO (MAS) RESPUESTAS]
END

TO HALLAR :INTERROGANTES :DATOS
   MAKE "VARS []
   MAKE "RESP []
   COMPARAR :INTERROGANTES :DATOS
   PRINTL :RESP
END
```

Ahora COMPARAR tiene que realizar una tarea bastante difícil. Tomemos como ejemplo la entrada [[MATO ZACARIAS ?UTENSILIO] [POSEE ?SOSPECHOSO ?UTENSILIO]]. COMPARAR revisa la base de datos, de hecho en hecho, en busca de una pareja para el primer interrogante, y acaba emparejando ?UTENSILIO con HACHA. La rutina considera entonces el segundo interrogante ([POSEE ?SOSPECHOSO ?UTENSILIO]), empezando otra vez por el comienzo de la base de datos. Se encuentra una pareja para la segunda condición, con el valor para ?UTENSILIO de HACHA y el de MATEO para ?SOSPECHOSO. No hay ningún otro interrogante, de modo que ésta es una posible solución.

Pero todavía no hemos terminado; puede haber otros valores que satisfagan el segundo interrogante, manteniendo a ?UTENSILIO como HACHA. De modo que comparar prosigue revisando la base de datos a partir del lugar en que la dejó, y realmente encuentra una segunda solución para ?SOSPECHOSO con JOSE. Por supuesto, el procedimiento no se detiene allí, sino que continúa buscando en la BASEDATOS. En esta ocasión llega hasta el final sin haber encontrado ningún otro valor emparejado.

Es posible, no obstante, que exista otra solución para el primer interrogante distinta de HACHA para ?UTENSILIO, de manera que hemos de volver atrás hasta el punto de la base de datos en el que encontramos la pareja y continuar desde allí. Este proceso se denomina *vuelta atrás (backtracking)*. En este



	3 ✗	1 ✗	4 sí	4 ✗	4 ✗	2 ✗	4 sí	3 ✗
	3 sí	2 ✗	1 ✗	3 ✗	3 ✗	2 ✗	3 ✗	3 sí
	2 ✗	2 sí	2 ✗	1 ✗	2 ✗	2 sí	2 ✗	2 ✗
	1 ✗	2 ✗	4 ✗	✓	✓	2 ✗	4 ✗	3 ✗
	3 ✗	2 ✗	4 ✗	✓				
	2 ✗	2 sí	2 ✗	2 ✗				
	3 ✗	2 ✗	4 sí	4 ✗				
	3 sí	2 ✗	3 ✗	3 ✗				

David Higham

Enigma de un crimen

El señor Harcourt fue hallado en la parte trasera de una furgoneta con 30 puñaladas hechas con un cincel. La policía identificó a cuatro sospechosos: un constructor, un carnicero, una jardinera y un diseñador. Cada uno de ellos tuvo acceso a instrumentos punzantes: un cuchillo (el carnicero), tijeras (la jardinera), un cortador (el diseñador) y un cincel (el constructor). Uno de ellos fue visto en una esquina, otro en el cobertizo de un jardín, y un tercero aseguró haber estado acostado, durmiendo. El que pueda ser relacionado con la furgoneta es el asesino. La investigación condujo a lo siguiente: **1)** La noche del crimen, ningún sospechoso estaba en posesión de su herramienta; **2)** Al carnicero se le vio en el cobertizo abriendo cartas con el cortador; **3)** Un testigo presencial confirmó que el constructor se hallaba parado en una esquina, donde la jardinera encontraría luego las tijeras que le faltaban; **4)** La jardinera estaba acostada, y utilizó un cuchillo de carnicero para prepararse un bocadillo

Complem. al LOGO

Algunas versiones de LOGO MIT no poseen EMPTY? ni MEMBER? Ya hemos ofrecido definiciones para ellas en anteriores *Complementos al* LOGO.
En todas las versiones LCSI utilice EMPTYP por EMPTY? y MEMBERP por MEMBER? Existe una primitiva, EQUALP, que comprueba si sus dos entradas son iguales. Empléela para comparar listas y palabras en lugar del signo de igualdad (que funciona para listas sólo en algunas versiones LCSI).
La sintaxis IF del LOGO LCSI queda demostrada de la siguiente manera:

```
IF EMPTYP :CONTENIDO
[PRINT[NADA ESPECIAL]]
[PRINT :CONTENIDO]
```

Si la condición es verdadera se lleva a cabo la primera lista después de la condición, y la segunda si es falsa.
El LOGO LCSI también admite la sintaxis TEST, IFTRUE, IFFALSE para las condiciones

caso, en realidad no existe ninguna otra solución.

Para no perder la pista del punto en que se encuentra en su asignación de variables, COMPARAR coloca los valores actuales en una *pila* antes de utilizar PAREJA? (puesto que PAREJA? podría alterar estos valores), y posteriormente los recupera. Éste es el procedimiento completo:

```
TO COMPARAR :INTERROGANTES :DATOS
   IF EMPTY? :INTERROGANTES THEN MAKE
      "RESP FPUT :VARS :RESP STOP
   IF EMPTY? :DATOS THEN STOP
   EMPUJAR :VARS
   TEST PAREJA? FIRST :INTERROGANTES FIRST
      :DATOS
   IFTRUE COMPARAR BUTFIRST :INTERROGANTES
      :BASEDATOS
   TIRAR "VARS
   COMPARAR :INTERROGANTES BUTFIRST :DATOS
END
```

En COMPARAR utilizamos una pila para seguir la pista del valor de VARS, en lugar de una variable temporal, debido a que COMPARAR se llamaría a sí misma entre el momento en que deseáramos guardar los valores y el momento en que deseáramos recuperarlos. Por consiguiente, cualquier variable temporal de este tipo sería destruida en la siguiente llamada y se perderían los valores originales. La pila impide que suceda esto.

EMPUJAR coloca un valor "encima" de la pila, creando primero la variable PILA si la misma no existiera aún.

```
TO EMPUJAR :DATOS
   IF NOT THING? :PILA THEN MAKE "PILA []
   MAKE "PILA FPUT :DATOS :PILA
END
```

TIRAR toma un elemento de la pila y lo asigna a una variable.

```
TO TIRAR :NOMBRE
   MAKE :NOMBRE FIRST :PILA
   MAKE "PILA BUTFIRST :PILA
END
```

Lo que tenemos ahora son los rudimentos de un lenguaje de "programación lógica". Éste es un lenguaje en el que simplemente se añaden hechos y reglas a una base de datos y luego se interroga a esa base de datos mediante descripciones lógicas de los datos que se requieren. Hasta la fecha, el mejor ejemplo de lenguaje de programación lógico es el PROLOG; ¡pero ésa es otra historia!



Paso a paso

Estudiaremos en este capítulo los motores paso a paso, usados de manera preferencial en el control de los movimientos de todo tipo de robots

La construcción de un motor paso a paso es muy diferente de la de un motor normal. Para comprender sus principios de funcionamiento, veremos en primer lugar cómo trabaja un motor paso a paso simplificado.

En nuestro ejemplo (véase el diagrama titulado "Un paso cada vez" en la página siguiente) hay dos juegos de bobinas ("a" y "b") en el estator y dos pares de polos electromagnéticos en el rotor. En los motores que utilizamos en la construcción de nuestro robot se incluye un mayor número de bobinas de estator y de polos de rotor que los que aparecen en el ejemplo.

El único problema de esta conveniente forma de control de motor es que el motor consume tanta corriente cuando está parado como cuando está en movimiento.

Por otra parte, no es posible hacer girar el motor a gran velocidad: las distintas bobinas no se pueden activar y desactivar con la rapidez suficiente. No obstante, ninguno de estos problemas tiene importancia en nuestra aplicación.

Nuestro motor simplificado sólo puede girar en pasos de 45°. Además, no se puede controlar la dirección de la rotación. Los motores utilizados en el robot, sin embargo, poseen cuatro juegos de bobinas que se activan por pares, y el rotor, asimismo, posee muchas más bobinas de las que muestra nuestro ejemplo. Ello significa que sí se puede controlar la dirección de rotación y que el ángulo de paso se reduce a 7,5°. Para conseguir este giro por pasos tan preciso, se deben activar las cuatro bobinas en una secuencia dada y compleja, del siguiente modo:

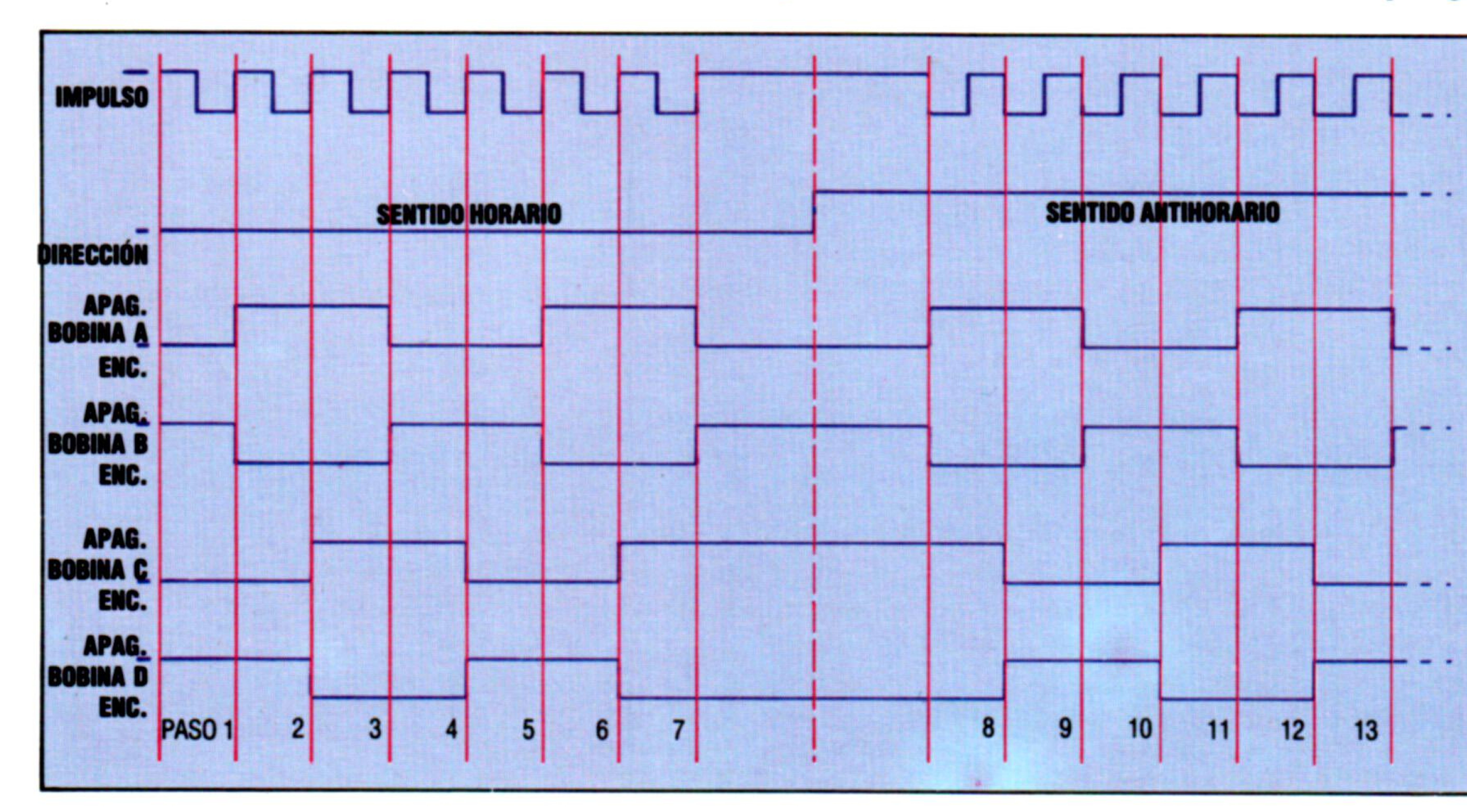
Tabla de la secuencia de activación

Paso	Bobina A	Bobina B	Bobina C	Bobina D
1	encendida	apagada	encendida	apagada
2	apagada	encendida	encendida	apagada
3	apagada	encendida	apagada	encendida
4	encendida	apagada	apagada	encendida
5	encendida	apagada	encendida	apagada

Esta secuencia de activación se podría gestionar mediante software, utilizando cuatro bits de la puerta para el usuario para controlar las cuatro bobinas. Sin embargo, ello requiere una programación complicada y, ciertamente, el BASIC no produciría estas secuencias de control con la rapidez suficiente. Un método más sencillo consiste en emplear un chip que ha sido diseñado especialmente para el control de motores paso a paso: el SAA 1027. Éste contiene los activadores de salida y todos los circuitos lógicos para activar las bobinas en el orden correcto necesario para activar un motor paso a paso.

Para hacer que el motor efectúe un giro de un paso, se requiere un único impulso proveniente de la puerta para el usuario, siendo necesaria aún otra línea de señal para determinar la dirección de la rotación. El chip contiene etapas de entrada para detectar los cambios en las tres entradas: un impulso para hacer que el motor gire un paso, una entrada de inicialización y una entrada de dirección que invierte la secuencia activadora de las bobinas del estator. Las entradas van a parar a un circuito contador bidireccional para producir la correcta secuencia de salidas hacia las bobinas del estator.

Por último, el chip también contiene una etapa activadora de salida de potencia que puede manipular hasta 550 mW. La inclusión de esta etapa sig-



Activando las bobinas

El diagrama muestra cómo las cuatro bobinas se energizan en secuencia, produciendo en el rotor un movimiento en sentido horario. Después de haberse completado los pasos 1, 2, 3 y 4 la secuencia se reinicia, activando el paso 5 la misma combinación de bobinas que el paso 1. Cuando cambia la señal de dirección, la secuencia se invierte. El paso 9 activa las bobinas en la misma combinación que el paso 7; el paso 10 energiza las bobinas de la misma forma que el paso 6, y así sucesivamente, haciendo que el rotor gire en sentido antihorario, siendo la posición del rotor después del paso 8 la dirección más extrema en el sentido horario

nifica que el motor se puede conectar directamente al chip sin que exista necesidad de transistores de potencia externos.

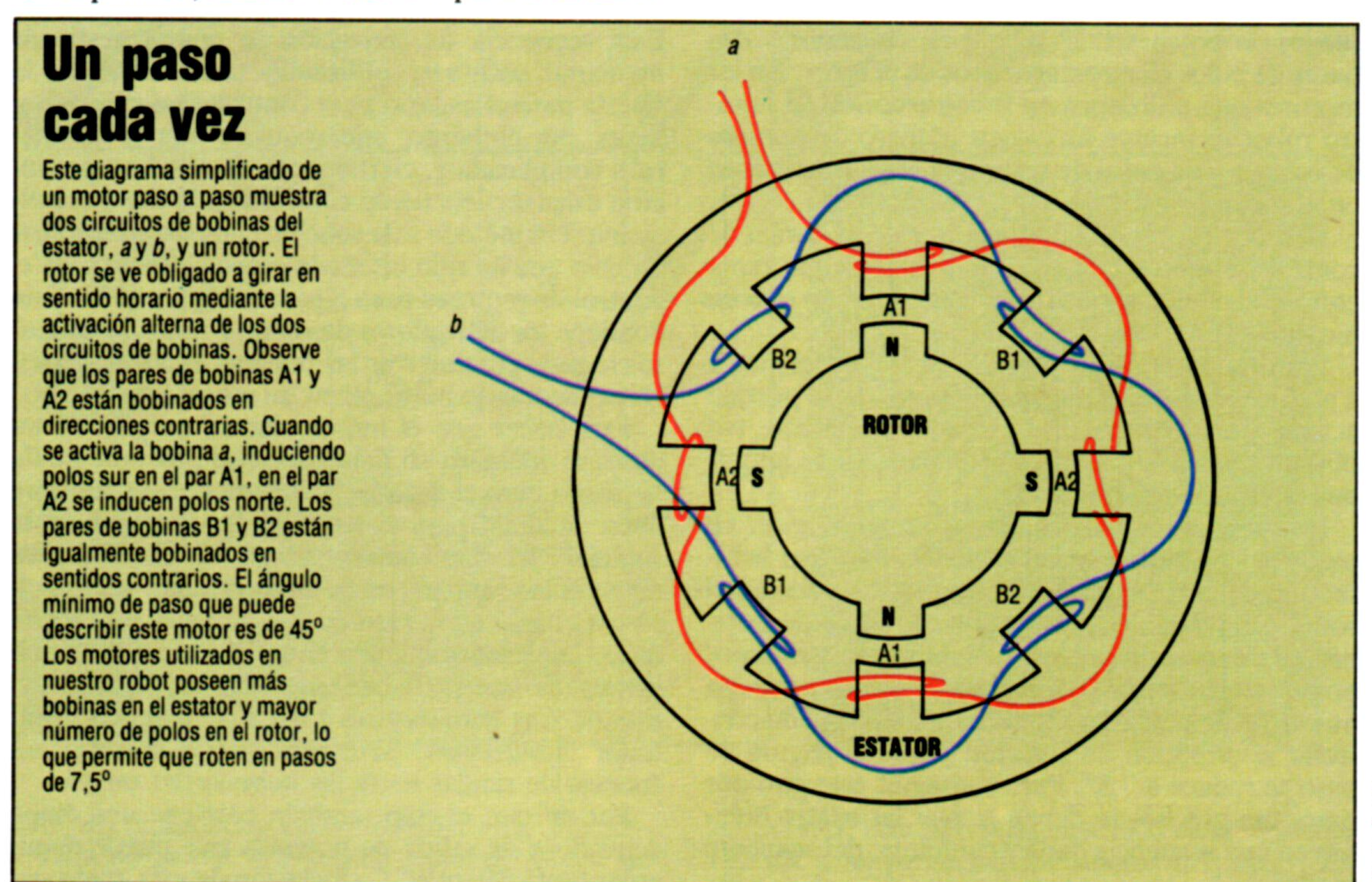
La complejidad del chip activador de motores paso a paso permite que el resto del circuito necesario para el control del robot sea verdaderamente muy simple. Cada motor requiere uno de estos chips, al cual se conecta el motor. Lamentablemente, los chips activadores operan a un voltaje de alrededor de 12 V, mientras que la puerta para el usuario de su ordenador opera a 5 V. Es decir, un 0 lógico es 0 V (más o menos) y un 1 lógico es 5 V. Las entradas del chip activador exigen 0 V para una entrada 0 y entre 7,5 y 12 V para un 1. Para la conexión en interface de la puerta para el usuario con los chips activadores también precisaremos un chip buffer especial de dos voltajes con las entradas operando en un voltaje y las salidas en otro. Éste es el chip 40109, también necesario para el circuito.

Lista de componentes

Cantidad	Artículo
1	Chip buffer 40109
3	Conector DIL de 16 patillas
2	Resistencia de 100 ohmios
2	Resistencia de 270 ohmios 0,5 W
2	Condensador de 0,1 μF
1	Condensador de 1000 μF 25 V
1	Veroboard de 24 franjas×50 agujeros
1	Rollo cable estañado 20 swg

PIEZAS DE RADIO

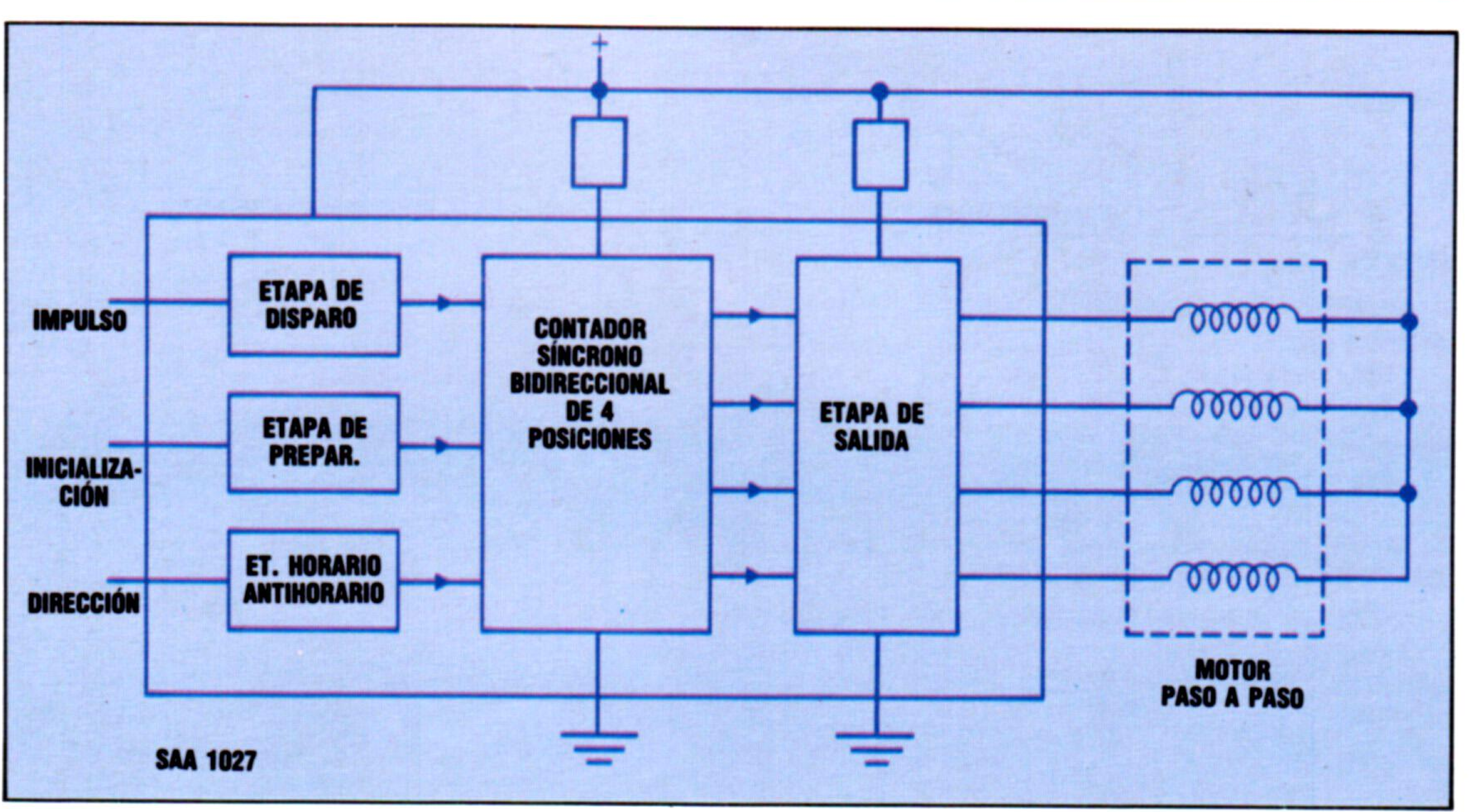
2	Activador de motor paso a paso SAA 1027



Un paso cada vez

Este diagrama simplificado de un motor paso a paso muestra dos circuitos de bobinas del estator, *a* y *b*, y un rotor. El rotor se ve obligado a girar en sentido horario mediante la activación alterna de los dos circuitos de bobinas. Observe que los pares de bobinas A1 y A2 están bobinados en direcciones contrarias. Cuando se activa la bobina *a*, induciendo polos sur en el par A1, en el par A2 se inducen polos norte. Los pares de bobinas B1 y B2 están igualmente bobinados en sentidos contrarios. El ángulo mínimo de paso que puede describir este motor es de 45°
Los motores utilizados en nuestro robot poseen más bobinas en el estator y mayor número de polos en el rotor, lo que permite que roten en pasos de 7,5°

Kevin Jones



La fuerza motriz
Si bien la lógica de los chips activadores de motores paso a paso es compleja, los principios de operación son fáciles de comprender. Con el fin de hacer girar el rotor, se deben activar las bobinas del estator de acuerdo a una determinada secuencia. Un contador bidireccional se mueve a través de esta secuencia de una etapa cada vez en respuesta a una señal de impulso. Los pasos de la secuencia también se pueden efectuar en la dirección opuesta si se cambia la entrada de la línea de dirección, haciendo que el motor gire en el sentido contrario. Una tercera entrada permite, si es necesario, devolver el rotor a la posición que ocupaba al comienzo de la secuencia

Liz Dixon



El diagrama del circuito
La circuitería necesaria para activar los dos motores paso a paso es directa, utilizándose dos chips activadores SAA 1027 para proporcionar la adecuada secuencia de activación de bobinas para cada motor. Dado que los chips activadores operan a 12 V y las señales de la puerta para el usuario de su micro son de sólo 5 V, se emplea un chip buffer adicional para aislar la circuitería de 12 V del ordenador y traducir las señales de la puerta para el usuario, de menor voltaje, a las señales de mayor voltaje que requieren los chips activadores. En el próximo capítulo le mostraremos cómo se conecta la placa de circuitos con los motores y el enchufe D, y cómo efectuar las conexiones correctas con la puerta para el usuario del ordenador

1000 µF
100 Ω 270 Ω
0,1 µF
DIR IMP INIC SAA 1027
MOTOR 1
40109
100 Ω 270 Ω
0,1 µF
DIR IMP INIC SAA 1027
MOTOR 2
+12 V
+5 V
D0
D1
D2
D3
D5 D4
D7 D6
TIERRA

CONDENS. 1000 µF
CHIP ACTIV. SAA 1027
CONDENS. 0,1 µF
D C E E B A
TIERRA
CHIP SAA ACTIVADOR 1027
D3
D2
RESISTENCIA 270 Ω
RESISTENCIA 100 Ω
D1
CHIP BUFFER 40109
D0
+5 V
+12 V
C D E E A B
RESIST. 100 Ω
RESIST. 270 Ω
CABLES DE ENLACE
CONECTOR DIL

C D E E A B
CORTES PISTAS
CHIP BUFFER 40109
+12 V
+5 V
D0
D1
D2
D3
CHIP ACTIV. SAA 1027
TIERRA
CORTE DE PISTAS
CHIP ACTIV. SAA 1027
D C E E B A

Kevin Jones

Construcción de la placa

- Corte la placa (tamaño 24 franjas × 35 agujeros) y efectúe los cortes de pistas que se indican en el diagrama.
- Suelde los tres conectores de chips y después los cables de enlace.
- Suelde las cuatro resistencias. Es indistinto colocarlas en la placa con una u otra orientación.
- Los dos condensadores de 0,1 µF también se pueden instalar con cualquier orientación, pero el condensador grande de 1000 µF se debe instalar con el terminal positivo del lado de la placa donde están los dos chips.
- Coloque los chips en su sitio, asegurándose de ponerlos del lado correcto. El extremo del chip que tiene una muesca debe quedar del lado de la placa donde están los dos conectores.
- Compruebe toda la placa. Verifique no sólo que los componentes estén colocados correctamente, sino también que no haya gotas de soldadura que hagan puente entre pistas adyacentes. Pase la punta de un cuchillo a lo largo del agujero entre cada pista para eliminar cualquier resto de soldadura.

Atando cabos

Concluimos nuestro curso sobre el lenguaje assembly del 6809. Nos queda sólo atar algunos cabos sueltos del programa depurador

El módulo principal como primer objetivo establecer el mecanismo de interrupción, lo que nos permite colocar puntos de ruptura en el programa a depurar. Éstos transfieren el control al programa depurador y nos facilitan el poder inspeccionar el contenido de los registros y de las posiciones de memoria. Debemos seguidamente obtener la dirección de inicio del programa a depurar para que el control pueda pasar a ella por medio de la orden S. El resto de la rutina principal se refiere a la obtención de órdenes desde el teclado y su ejecución; el control se transfiere al programa a depurar por medio de las órdenes S y G y se devuelve al programa depurador por medio de las instrucciones SWI insertadas en los puntos de ruptura.

En la lección anterior ya se codificaron dos fases de inicialización para este módulo (véase p. 1297). El punto de entrada de las interrupciones sigue inmediatamente a la llamada de la subrutina. Aquí la primera instrucción S guardará (S*ave*) el puntero de la pila pra que pueda ser usado para referenciar los valores de los registros guardados en la pila por la SWI. La siguiente fase es la interpretación de la orden. Ya hemos elaborado rutinas para producir todas las órdenes, con lo cual el problema se reduce aquí a seleccionar la subrutina apropiada para la orden entrada.

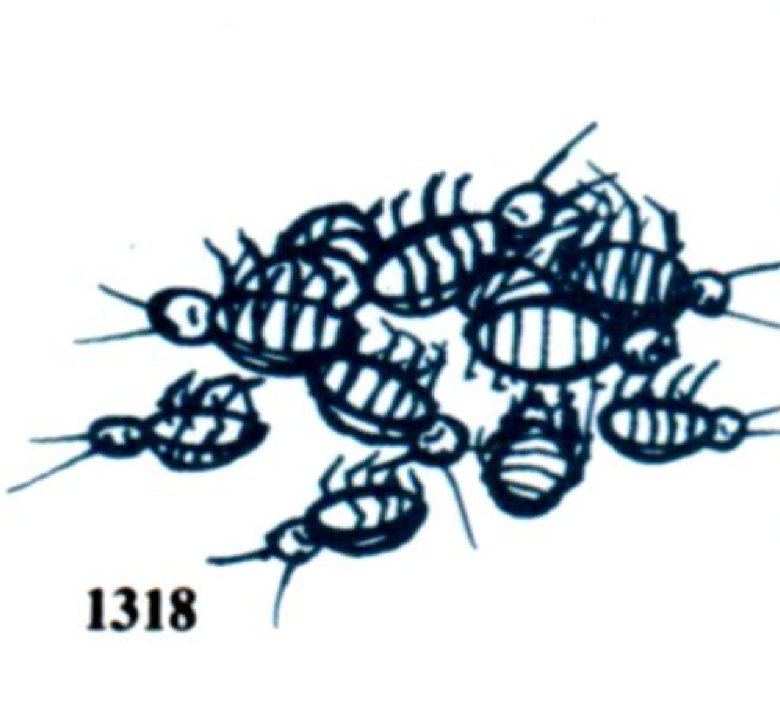
Uso de la tabla de salto

Es posible codificar esto con algo semejante a un conjunto de IF anidados, pero preferimos aprovechar la circunstancia de que la rutina Tomar-Orden devuelve un desplazamiento a la tabla de caracteres de órdenes y nos limitaremos a realizar estas llamadas empleando la tabla de salto. Quizá no sea éste el método más eficaz para el caso, pero es una técnica útil que vale la pena conocer. Requiere el establecimiento de una tabla de direcciones para cada una de las subrutinas que tratan una orden.

La instrucción JMP, a diferencia de las instrucciones de bifurcación, puede servirse de cualquiera de los modos normales de direccionamiento, incluidos el indexado y el indirecto. Si cargamos X con la dirección base de la tabla y empleamos el desplazamiento contenido en B (duplicado, ya que será una tabla de direcciones de 16 bits, no como la tabla de letras de órdenes que es de 8 bits), ocurrirá que la orden

```
JMP [B,X]
```

transferirá el control a la rutina apropiada. La llamada BSR se realiza a la dirección de esta instrucción de salto. Puesto que necesitamos establecer esta tabla por anticipado, es preciso añadir otra fase en la inicialización para llevar a cabo esta operación.



Proceso Est.-Tabla-Saltos

es una tabla de 8 direcciones de 16 bits
etc., son las direcciones de inicio de las subruunas de órdenes (*commands*)

```
FOR (para) cada subrutina
   Tomar dirección de inicio
   Guardar dirección de inicio en Tabla-Saltos
Endfor
```

Debemos ahora ocuparnos de lo que pasa al final de la ejecución, en el momento en que aparece la orden Q, abandonar (*quit*), aunque en realidad hay muy poco que hacer. Parece lógico dejar intactos tanto el depurador como el programa depurado, de modo que puedan ser ejecutados de nuevo si fuera necesario.

Al finalizar, la pila ha de quedar en la misma situación que tenía cuando comenzamos. Una solución pudiera ser el empleo de otra pila independiente para nuestro programa poniendo S a un valor nuevo y después restaurar el antiguo. Es una técnica generalmente útil, pero en nuestro caso puede resultar difícil encontrar un espacio libre en la memoria, teniendo el depurador colocado encima de otro programa. Una segunda solución consiste en incrementar sencillamente S en una cantidad adecuada para descartar lo que hayamos dejado allí, pero resulta difícil porque desconocemos si ha sucedido o no una interrupción y las cantidades en la pila serían diferentes. La solución más inmediata es guardar el valor inicial de S y restablecerlo como última operación del programa.

El mecanismo de interrupción, según se establece en el procedimiento de inicialización, almacena tres bytes en la dirección dada por el vector SWI en $FFFA; debemos almacenarla si no queremos obtener resultados desconcertantes en el momento en que el sistema operativo emplee SWI para sus operaciones. Necesitamos, pues, otra fase más de inicialización en la que guardemos estos valores que han de ser restaurados en nuestra rutina de abandono.

Proceso Guardar-Valores

Data:
Guarda se compone de cinco bytes que almacenarán los valores a salvar
Puntero-Pila es el valor en curso de S increm. en 2
Vector-SWI se encuentra en $FFFA
Proceso:
Guardar Puntero-Pila en Guarda
Tomar Vector-SWI
Guardar tres bytes de Vector-SWI en Guarda



La rutina de abandono (orden Q) no tiene más que invertir este proceso y devolver el control al sistema operativo. Lo cual puede realizarse de varias maneras: la instrucción SWI puede servir para ello, una vez restablecido su vector; o bien, se puede hacer un salto a un punto conocido de entrada en el sistema operativo. Es posible un salto a través del vector de reinicio en $FFFE, para devolver el control al sistema operativo, aunque esto puede provocar una inicialización de todo el sistema.

Proceso Abandono

Data:

Guarda son cinco bytes para almacenar los valores a salvar
Puntero-Pila es el valor en curso de S, increm. en 2
Vector-SWI está en $FFFA
Vector-Reinicio está en $FFFE

Proceso:

Restaurar en Vector-SWI tres bytes de Guarda

Flujo del programa
Los presentes diagramas de flujo corresponden a los módulos del programa depurador de errores. Se han colocado según el orden en que son llamados por las restantes subrutinas. Dentro de los diagramas, las celdillas ribeteadas de azul indican rutinas individuales a las que se llama

Restaurar Puntero-Pila
Salto al sistema operativo

Ya estamos preparados para codificar el módulo principal. Muy poco ha cambiado el diseño desde que se esbozó en un principio, y esencialmente sigue siendo el mismo.

Módulo principal

Data:
- **Interrogación** para entrada de la orden, es el carácter '>' en ASCII
- **Desplazamiento-Orden** a la tabla de caracteres de órdenes y Tabla-Saltos

Proceso:

```
Guardar-Valores
Establecer-Tabla-Saltos
Establecer-Interrupciones
Tomar-Dirección-Inicio
REPEAT
   Visualizar-Interrogación
   Tomar-Orden
   Ejecutar-Orden
INDEFINITELY
```

Con ello concluye nuestro programa depurador de errores. Parece que nos salió troceado en exceso, pero la fragmentación es típica de la programación modular. En este punto podríamos optimizar la codificación si nos dedicáramos a buscar atajos. Por ejemplo, se habrá dado cuenta de que hemos barajado una gran cantidad de valores sólo para cerciorarnos de que se encontraban en el registro adecuado para una subrutina; si ahora redefine el empleo de los registros, ciertamente ahorrará pasos reiterativos. Pero no se lo aconsejamos si el espacio de la memoria con que cuenta no es limitado. Según se van necesitando, hemos definido en bastantes sitios diferentes las mismas áreas de datos. Dos son las maneras de manejar las áreas de datos en el programa completo: se pueden retener los datos junto con el módulo que los emplea, que es la mejor opción teórica; o bien pueden definirse todos los datos juntos al comienzo del programa, lo cual tiene sus ventajas reales cuando se desea utilizar un desensamblador (o incluso un depurador) para el programa.

El depurador debe ser cargado en cualquier espacio libre de la memoria que el programa a depurar no ocupe o emplee. Se le da entrada mediante un salto al punto de entrada DEBUG (depurar); es, por tanto, necesario conocer su dirección antes de comenzar.

Establecer Tabla Salto

```
JTABLE   RMB    16            Espacio para ocho direcciones de dos bytes
SETUPJ   LEAY   JTABLE, PCR   Dir. base de tabla en Y
         LEAX   CMDB,PCR      Dirección inicio de la subrutina CMDB
         STX    ,Y++          La almacena en tabla
         LEAX   CMDU,PCR      Dirección inicio de subrutina CMDU
         STX    ,Y++          La almacena en tabla
         LEAX   CMDD,PCR      Dirección inicio de subrutina CMDD
         STX    ,Y++          La almacena en tabla
         LEAX   CMDS,PCR      Dirección inicio subrutina CMDS
         STX    ,Y++          La almacena en tabla
         LEAX   CMDG,PCR      Dirección inicio subrutina CMDG
         STX    ,Y++          La almacena en tabla
         LEAX   CMDR,PCR      Dirección inicio subrutina CMDR
         STX    .Y++          La almacena en tabla
         LEAX   CMDM,PCR      Dirección inicio subrutina CMDM
         STX    ,Y++          La almacena en tabla
         LEAX   CMDQ,PCR      Dirección inicio subrutina CMDQ
         STX    ,Y++          La almacena en tabla
```

He aquí el salto efectivo a la subrutina. Asumimos que X contiene la dirección de JTABLE (Tabla Salto) y B del desplazamiento

```
DOCMD    JMP    [B,X]
```

Guardar Valores

```
SAVED    RMB    5             Cinco bytes por guardar
SAVEIT   LEAX   SAVED,PCR     Toma dir. para guardarla
         TFR    S,D           Transfiere S a D
         ADDD   #2            Suma dos en atención a la dirección de retorno
         STD    ,X++          Lo guarda
         LDY    $FFFA         Toma dir. de vector inter.
         LDA    ,Y+           Toma el primer byte a guardar
         STA    ,X+           Lo guarda
         LDD    ,Y            Toma otros dos bytes
         STD    ,X            Los guarda
         RTS
```

Orden Q

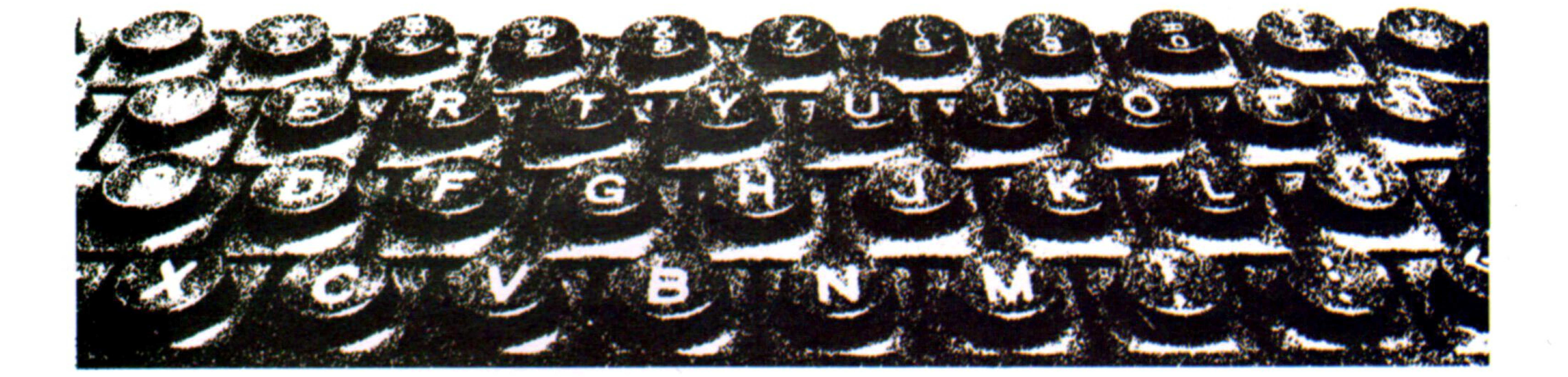
```
CMDQ     LEAX   SAVED,PCR     Dirección de Guarda
         LDY    $FFFA         Vector-SWI
         LDA    2.X           Primero de tres bytes
         STA    ,Y+           Restaurado
         LDD    3,X           Otros dos bytes
         STD    ,Y            Restaurados
         LDS    ,X            Puntero-Pila guardado
         JMP    [$FFFE]       Salto indir. por medio vector reinicio
```

Módulo principal

```
PROMPT   FCB    ,>
STACKP   RMB    2             Puntero-Pila para Visualizar-Registros
DEBUG    BSR    SAVEIT        Guardar-Valores
         BSR    SETUPJ        Establecer-Tabla-Salto
         BSR    INIT          Establecer-Interrupción y Tomar-Dirección-Inicio
ENTRY    STS    STACKP,PCR    Guardar Puntero-Pila
         LEAX   JTABLE,PCR
REPT02   LDA    PROMPT,PCR    Tomar Interrogación y visualizarlo
         BSR    OUTCH
         BSR    GETCOM        Tomar Orden
         LSLB                 Doble despl. para tabla 16 bits
         BSR    DOCMD         Obedecer Orden
         BRA    REPT02        Siguiente Orden
```

mi computer

Editorial Delta, S.A.



10066

9 788485 822836